

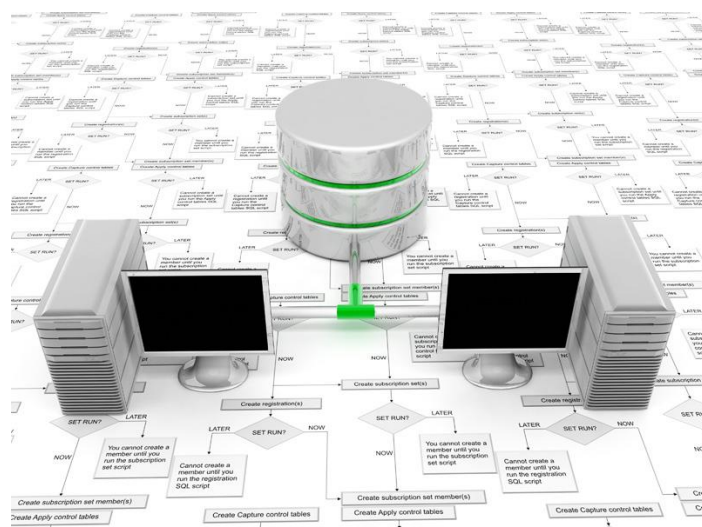
**КАЗАНСКИЙ (ПРИВОЛЖСКИЙ) ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ**  
**ИНСТИТУТ ВЫЧИСЛИТЕЛЬНОЙ МАТЕМАТИКИ**  
**И ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ**

**Кафедра системного анализа и информационных технологий**

**А.А. АНДРИАНОВА, Т.М. МУХТАРОВА, Р.Г. РУБЦОВА**

**ЛАБОРАТОРНЫЙ ПРАКТИКУМ**  
**ПО КУРСУ**  
**«ТЕХНОЛОГИИ БАЗ ДАННЫХ»**

**Учебное пособие**



**Казань – 2016**

**УДК 004.43**  
**ББК 32.973.26 – 018.1**

*Принято на заседании кафедры системного анализа  
и информационных технологий  
Протокол № 5 от 2 февраля 2016 года*

*Принято на заседании учебно-методической комиссии Института  
Вычислительной математики и информационных технологий  
Протокол № 6 от 18 февраля 2016 года*

**Рецензенты:**

кандидат педагогических наук, доцент кафедры  
информационных систем КФУ **Е.Е.Лаврентьева**;  
кандидат физико-математических наук,  
доцент кафедры анализа данных и исследования операций КФУ **В.В.Бандеров**

**Андрианова А.А., Мухтарова Т.М., Рубцова Р.Г.**

**Лабораторный практикум по курсу «Технологии баз данных»:**

**Учебное пособие** / А.А. Андрианова, Т.М. Мухтарова, Р.Г. Рубцова. –  
Казань: КФУ, 2016. – 97 с.

Учебное пособие предназначено для проведения лабораторного практикума по курсам «Технологии баз данных», «Базы данных» для студентов, обучающихся по направлениям «Фундаментальная информатика и информационные технологии», «Прикладная математика и информатика», «Информационная безопасность». Основная цель практикума – в сжатом изложении показать все основные задачи и фазы работы программиста с базами данных, технологии использования баз данных при проектировании приложений прикладной направленности. В пособии рассматриваются примеры с помощью нескольких популярных систем управления базами данных (СУБД).

© **Андрианова А.А.**

**Мухтарова Т.М.**

**Рубцова Р.Г., 2016**

© **Казанский университет, 2016**

## СОДЕРЖАНИЕ

ВВЕДЕНИЕ .....	4
ЧАСТЬ I. СЕРВЕРНЫЕ ТЕХНОЛОГИИ .....	11
1.1. МОДЕЛЬ ДАННЫХ.....	11
1.2. ПЕРЕНОС БАЗЫ ДАННЫХ НА ДРУГОЙ СЕРВЕР .....	19
1.3. КОМАНДЫ МОДИФИКАЦИИ ДАННЫХ (DML) .....	37
1.4. ВЫБОРКА ДАННЫХ. ОПЕРАТОР SELECT (DQL) .....	41
1.5. ХРАНИМЫЕ ПРОЦЕДУРЫ. ФУНКЦИИ И ТРИГГЕРЫ .....	48
Часть II. КЛИЕНТСКИЕ ТЕХНОЛОГИИ .....	62
2.1. ВЫПОЛНЕНИЕ ЗАПРОСА К БАЗЕ ДАННЫХ ИЗ КЛИЕНТСКОГО ПРИЛОЖЕНИЯ.....	62
2.2. ПАРАМЕТРЫ ЗАПРОСА .....	67
2.3. ВЫПОЛНЕНИЕ КОМАНД DML. ....	68
2.4. ПОНЯТИЕ НАБОРА ДАННЫХ КАК ВИРТУАЛЬНОЙ БАЗЫ ДАННЫХ.....	69
2.5. СВЯЗЬ НАБОРА ДАННЫХ И БАЗЫ ДАННЫХ .....	70
2.6. КАК СИНХРОНИЗИРОВАТЬ ИЗМЕНЕНИЯ В НАБОРЕ ДАННЫХ С БАЗОЙ ДАННЫХ.....	71
2.7. ПОЛЬЗОВАТЕЛЬСКИЙ ИНТЕРФЕЙС НА ОСНОВЕ ТАБЛИЦ .....	72
2.8. ПОЛЬЗОВАТЕЛЬСКИЙ ИНТЕРФЕЙС НА ОСНОВЕ ОДНОЙ ЗАПИСИ.....	74
2.9. ГЕНЕРАЦИЯ ОТЧЕТОВ И ПЕЧАТНЫХ ФОРМ .....	76
2.10. ГЕНЕРАЦИЯ ОТЧЕТОВ В ФОРМАТЕ XML.....	78
ЧАСТЬ III. ВВЕДЕНИЕ В ХРАНИЛИЩА ДАННЫХ.....	81
3.1. ПРОЕКТИРОВАНИЕ ХРАНИЛИЩ ДАННЫХ .....	82
3.2. ЗАГРУЗКА ДАННЫХ .....	83
3.3. ПОИСК ИНФОРМАЦИИ В ХРАНИЛИЩЕ .....	89
3.4. ПОСТРОЕНИЕ ОТЧЕТОВ С ПОМОЩЬЮ ЗАПРОСОВ К ХРАНИЛИЩУ .....	95
СПИСОК ЛИТЕРАТУРЫ.....	97

## ВВЕДЕНИЕ

Базы данных являются одной из основных составляющих большинства современных приложений, особенно прикладного или аналитического характера. Любое предприятие имеет свою базу данных (а, возможно, и множество баз данных). Заходя в интернет, мы видим информацию из баз данных через сервисы социальных сетей, интернет-магазинов, электронных университетов и др. Немало математических задач связано с использованием баз данных. Примером тому являются задачи анализа данных или машинного обучения. Таким образом, знание и навыки работы с базами данных становятся неотъемлемой составляющей компетенции современного ИТ-специалиста. Данное учебно-методическое пособие призвано помочь студенту в практической форме приобрести необходимые навыки работы с базами данных и их использованием в различных приложениях.

Разработка приложения, использующего базу данных, включает в себя множество задач. Во-первых, требуется сформировать логическую модель базы данных и, как следствие, набор таблиц, которые будут хранить данные. Вторым моментом является выбор системы управления базами данных (СУБД), на котором будет храниться база. Именно СУБД отвечают за выполнение основных операций, выполняемых с базой данных. Во многом этот выбор зависит от масштабов создаваемого приложения. В дальнейшем следует определить серверную часть приложения, включающую определения целостности данных, серверные процедуры, позволяющие выполнять основные преобразования данных. Только после решения всех этих вопросов речь заходит о клиентской части приложения работы с базой данных. Некоторые СУБД имеют собственные средства создания клиентской части (например, MS FoxPro или более популярный MS Access), но в большинстве своем современные СУБД являются серверными, т.е. предоставляют средства доступа к данным из других приложений. Этот момент позволяет создавать гибкий пользовательский интерфейс на тех технологиях, которые являются более приемлемыми для пользователя. Отдель-

ным вопросом функционирования приложения базы данных являются вопросы экспорта и импорта данных из других источников информации и агрегация информации из различных источников для предоставления сводной и аналитической отчетности (концепция хранилищ данных).

Пособие создано в поддержку практикума по курсу «Технологии баз данных», который реализуется в Казанском (Приволжском) федеральном университете на кафедре системного анализа и информационных технологий. За время практикума каждый студент должен разработать собственное приложение баз данных, которое обязательно должно включать следующие элементы:

1. Создание логической модели базы данных. Описание ER-модели, генерация на ее основе реляционной модели данных.
2. Реализация модели в СУБД. В качестве СУБД могут быть выбраны: MS SQL Server, MySQL или PostgreSQL или иное серверное СУБД.
3. Заполнение базы данных.
4. Создание различных запросов на получение данных (для формирования навыков работы с реализацией различных операций реляционной алгебры). Для каждой из операций (исключая деление) нужно показать минимум три запроса (хотя один и тот же запрос может демонстрировать выполнение нескольких операций).
5. Создание хранимых процедур и триггеров для обеспечения серверной части работы с данными.
6. Создание клиентского windows-приложения для работы с базой данных. Приложение должно иметь возможности добавления, изменения и удаления информации.
7. Создание модулей экспорта и импорта информации в базу данных (интеграция с xml-файлами).
8. Реализация концепции хранилищ данных на примере создания OLAP-куба для многомерного поиска данных для публикации в отчетах.

Каждая из перечисленных задач рассматривается в учебно-методическом пособии на примере создания элементов приложения «Деканат», с помощью

которого предоставляются возможности отслеживать оценки, которые получают студенты во время сессии. Структурно в учебно-методическом пособии будет выделено три главы, посвященные разработке серверных средств (базы данных и серверных процедур), разработке клиентской части приложения и введению в концепции хранилищ данных.

В качестве средств разработки (программного обеспечения) нужно выбрать сервер баз данных, т.е. СУБД, инструментальную оболочку для работы с выбранным сервером, технологию создания клиентского интерфейса.

В качестве сервера баз данных можно использовать:

- MS SQL Server – устанавливается вместе с MS Visual Studio, которая может использоваться как оболочка доступа к базам данных. При установке SQL Server'у присваивается определенное имя, по которому к нему можно будет обращаться (по умолчанию SQLEXPRESS). Для локальной работы с сервером можно использовать при подключении имя (local). Свободной оболочкой (для некоммерческого использования) для MS SQL Server является программный продукт dbForge Studio компании DEVART (<http://www.devart.com/ru/dbforge/sql/studio>):

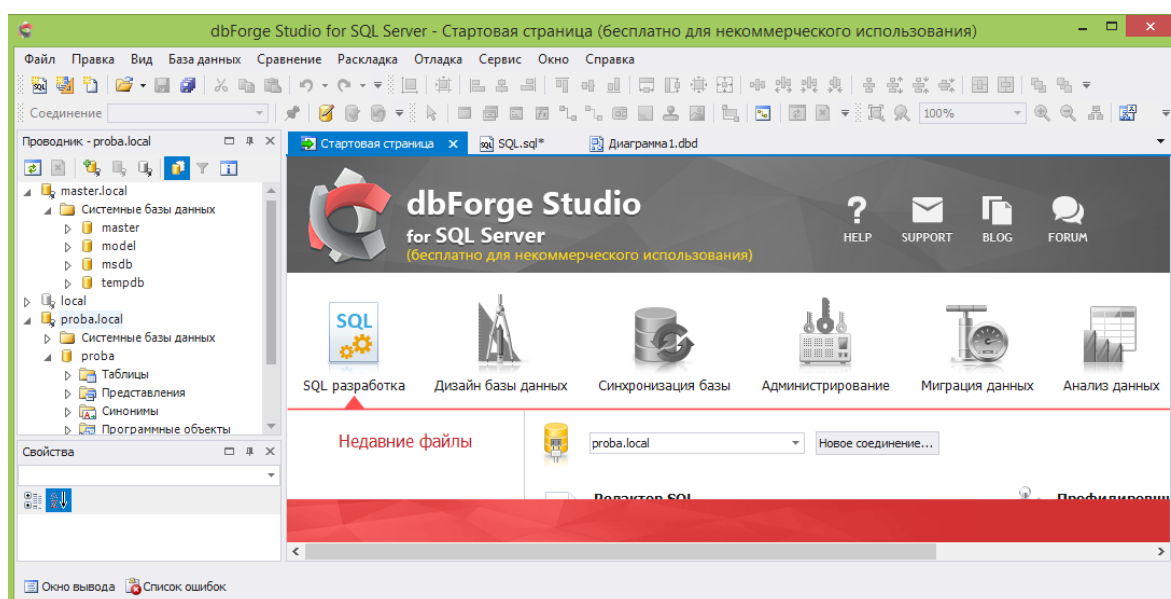


Рис. 1. Главное окно dbForge Studio для MS SQL Server.

Создание соединения оболочки с сервером производится с помощью меню «База данных» -> «Новое подключение...». Здесь вводятся параметры подключения и имя, по которому в дальнейшем к этому подключению можно будет обращаться:

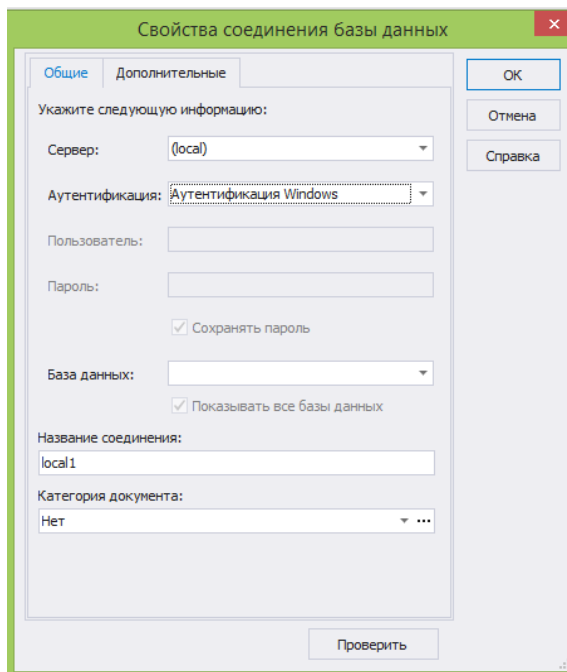


Рис. 2. Параметры соединения с базой данных MS SQL Server.

- MySQL (версии с 5.0). Этот бесплатный сервер баз данных устанавливается отдельно и конфигурируется с помощью специального wizard'а. Обратим внимание не то, что при конфигурировании экземпляра сервера требуется установить параметры учетной записи. По умолчанию, логин и пароль для сервера root. В качестве оболочки для работы с сервером MySQL можно использовать программный пакет MySQL Workbench – это свободное программное обеспечение, которое содержит средства моделирования, администрирования сервера и визуальной работы с базами данных, размещенными на нем.

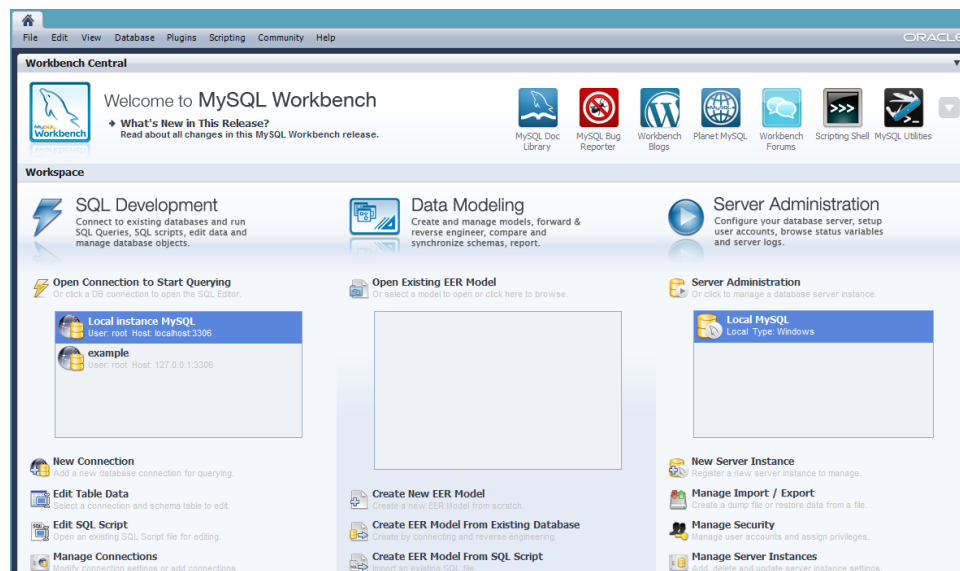


Рис. 3. Главное окно MySQL Workbench.

Для MySQL (аналогично MS SQL Server) компанией DEVART была разработана версия оболочки проектирования dbForge Studio. Она также является свободной для некоммерческого использования (<http://www.devart.com/ru/dbforge/mysql/studio>):

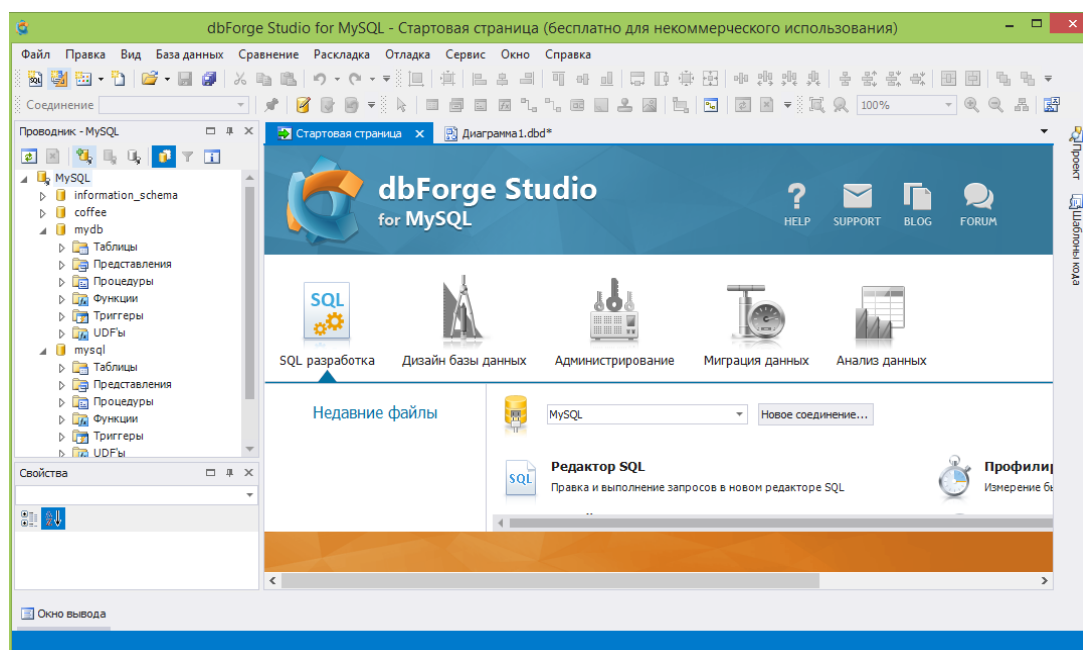


Рис. 4. Главное окно dbForge Studio для MySQL.

При создании подключения к MySQL серверу требуется указать другие параметры – это имя хоста, на котором установлен сервер баз данных (для локальных машин localhost), номер порта (по умолчанию MySQL ставится на порт 3306), логин и пароль учетной записи пользователя, а также



имя подключения. Еще не следует забывать на вкладке «Дополнительно» установить кодировку данных (сейчас настройки наиболее часто используют кодировку utf8) (MySQL очень чувствителен к кодировкам и отсутствие настройки кодировки может привести к проблемам с данными, написанными кириллицей):

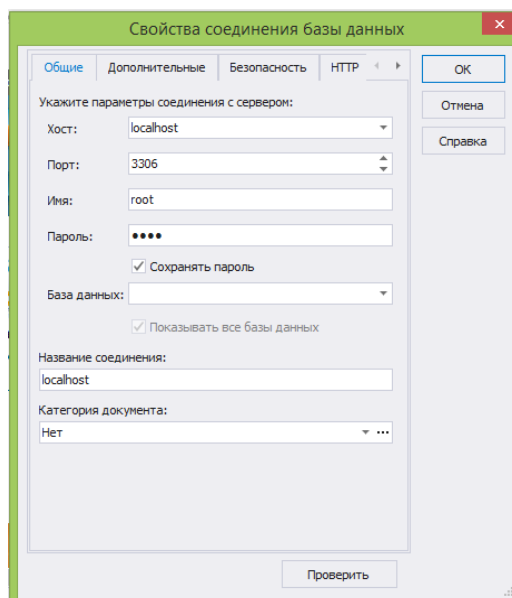


Рис. 5. Параметры соединения с базой данных MySQL.

- PostgreSQL также является свободным сервером баз данных. Также имеет оболочку проектирования pgAdmin. Существует уже оболочка dbForge Studio для PostgreSQL, однако на момент написания данного текста она была платным программным обеспечением.

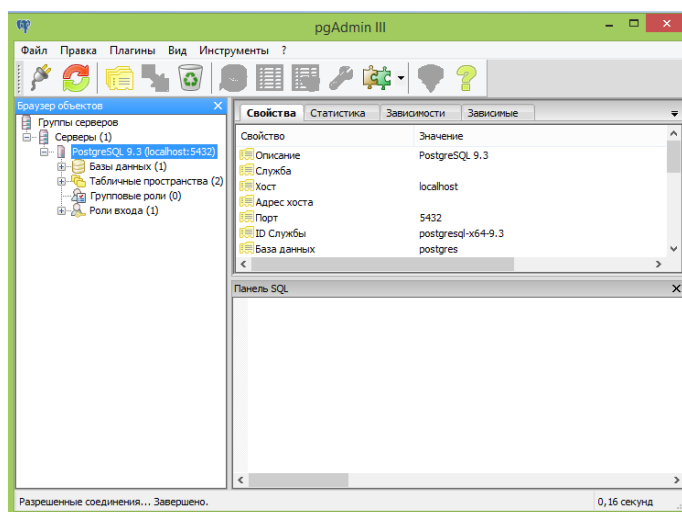


Рис. 6. Окно программы pgAdmin.

При установке сервера PostgreSQL и его дополнительного программного обеспечения будут запрошены параметры учетной записи пользователя. По умолчанию создается запись с логином postgres, пароль к которой устанавливает пользователь в момент установки. Аналогично MySQL, PostgreSQL идентифицируется хостом и номером порта (по умолчанию, 5432).

## ЧАСТЬ I. СЕРВЕРНЫЕ ТЕХНОЛОГИИ

### 1.1. МОДЕЛЬ ДАННЫХ

Разберем принципы формирования модели базы данных на примере приложения «Деканат». Модель будет создаваться с помощью инструментов моделирования данных в различных оболочках.

**Описание задачи.** Пусть требуется хранить и управлять информацией о результатах обучения студентов: об учебных группах; студентах, обучающихся в этих группах; дисциплинах, которые изучаются и сдаются в разные семестры; преподавателях, которые ведут эти дисциплины; оценках, которые были получены студентами при сдаче зачетов/экзаменов.

Существует несколько концепций моделей баз данных (иерархическая, сетевая, объектная, реляционная). Наиболее распространенной моделью является реляционная модель, которая очень тесно переплетается с принципами объектно-ориентированного анализа и еще одного популярного подхода в моделировании данных – ER-модели (модель «сущность-связь»).

ER-модель удобна для начального проектирования, поскольку она интуитивно понятна большинству пользователей. В ней выделяются понятия сущности (основные объекты базы), атрибуты (свойства сущности) и связи (взаимодействия между сущностями). В ряде оболочек именно в этих терминах и создан сервис создания модели данных.

Реляционная модель представляет всю базу данных как набор связанных таблиц. Большинство таблиц отвечает за хранение информации о сущностях (столбцы таблиц характеризуют их атрибуты). Среди атрибутов сущности выделяют ключевые атрибуты – атрибуты, которые являются идентифицирующими, точно определяющими запись, объект сущности. С помощью внедрения ключевых атрибутов одних сущностей (родительские таблицы) в качестве столбцов в другие таблицы (дочерние) реализуются различные связи между сущностями.

## Построение модели с помощью оболочки MySQL Workbench (версия 5.2.39 CE).

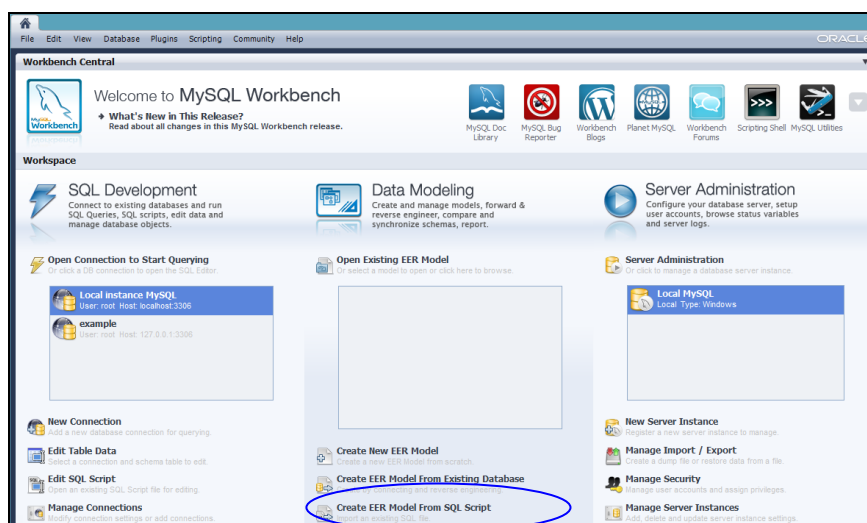


Рис. 7. Создание модели базы данных в MySQL Workbench.

Создаем новую ER-модель и диаграмму в модели. В полученном окне модели представлено полотно, на которое можно наносить новые таблицы, с помощью визуальных средств редактирования, создать столбцы (атрибуты) таблиц и с помощью панели инструментов создать связи между таблицами. При установке связи ключевые поля родительских сущностей добавляются в дочерние таблицы автоматически.

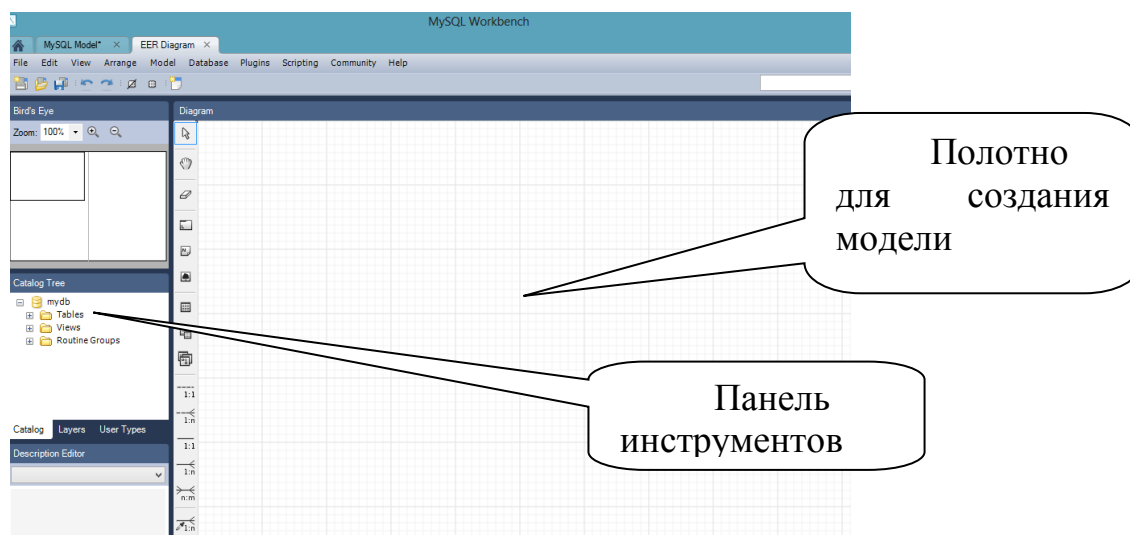


Рис. 8. Вид окна редактирования модели данных.

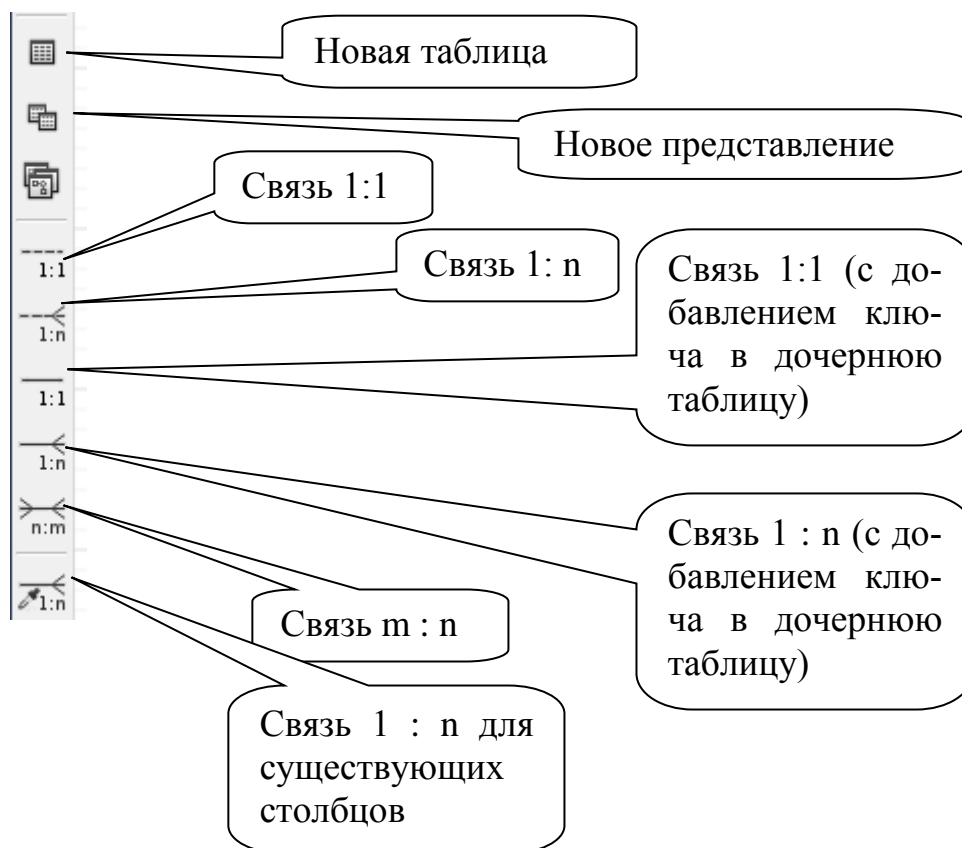


Рис. 9. Состав панели инструментов окна редактирования модели данных.

Проведем анализ состава таблиц для решаемой задачи. При описании столбцов таблицы поля, входящие в первичный ключ, будут подчеркнуты.

Имеется таблица **Студенты (Students):** (№Зач.книжки, ФИОСтудента, №Группы).

Для хранения групп не будем выделять отдельную таблицу.

Имеется таблица **Преподаватель (Teachers):** (№Преподавателя, ФИОПреподавателя, Должность, №Кафедры).

Чтобы избежать дублирования информации с названием кафедры введем справочную таблицу кафедр: таблица **Кафедра (Departments):** (№Кафедры, Название, Телефон).

Имеется таблица учебных дисциплин **Дисциплина (Subjects):** (№Дисциплины, Название).

Таблица **Сессия** содержит информацию о том, каков состав зачетов и экзаменов для каждой конкретной группы по семестрам, каким преподавате-

лям следует сдавать зачеты и экзамены: **Sessions** (№Группы, №Семестра, №Дисциплины, Отчетность, №Преподавателя). Заметим, что отчетность может определяться номером дисциплины и номером семестра, но в предположении наличии нескольких специальностей один и тот же предмет может сдаваться в разных семестрах разными группами. Поэтому отчетность и преподаватель зависят и от группы тоже.

Наконец, результаты сдачи сессии хранятся в таблице результатов **Results** (№Студента, №Группы, №Семестра, №Дисциплины, Баллы, ДатаСдачи, Оценка). Окончательную оценку хранить не требуется, так как она определяется количеством набранных баллов и таблицей оценок.

**Marks** (Оценка, НижняяГраница, ВерхняяГраница) – эта таблица является справочной и не связана с основными таблицами базы. Ее роль заключается в определении правильной оценки по набранным баллам.

В результате данного анализа задачи получится следующая модель:

- сначала формируется состав таблиц без связующих атрибутов:

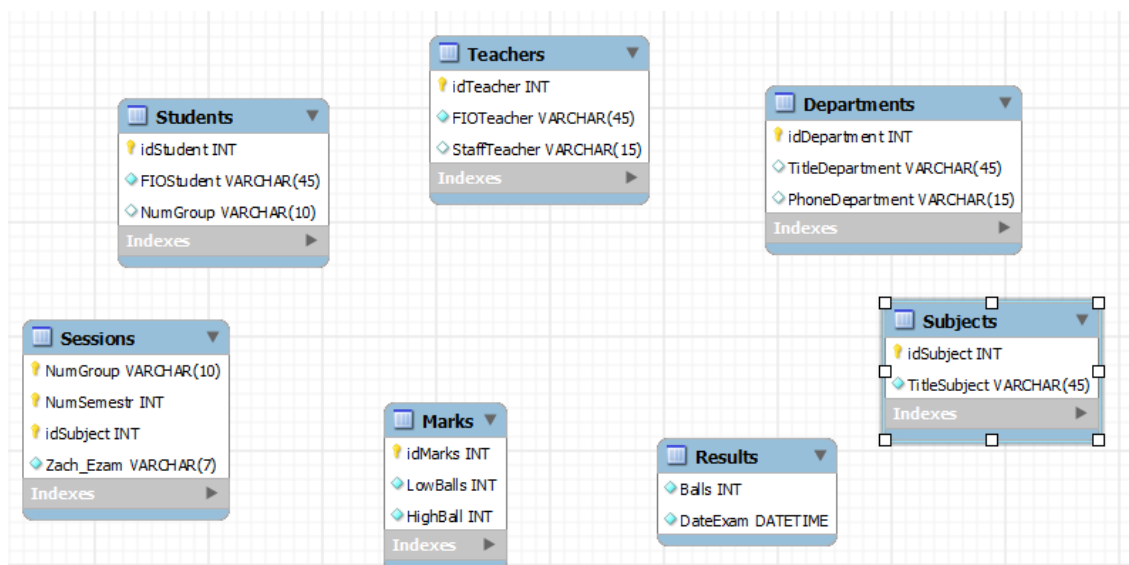


Рис. 10. Модель таблиц базы данных «Деканат» без указания связей.

- затем устанавливаем связи. Заметим, что можно было бы все связующие атрибуты сразу добавить в таблицы. Тогда все связи можно было бы добавить как связи «один-ко-многим» для существующих столбцов. Отметим также, что связь таблицы результатов и сессии не является очевидной, так как сессия зави-

сит от номера группы, а в таблице результатов указываются оценки конкретных студентов. Поэтому эту связь можно сделать идентифицирующей, а потом удалить из таблицы результатов атрибут номера группы. Другой вариант решения этой проблемы, добавить все поля в таблицу результатов и не устанавливать связь на уровне модели. Далее после создания таблиц в базе данных добавить ограничения внешних ключей для полей номера дисциплины и номера преподавателя.

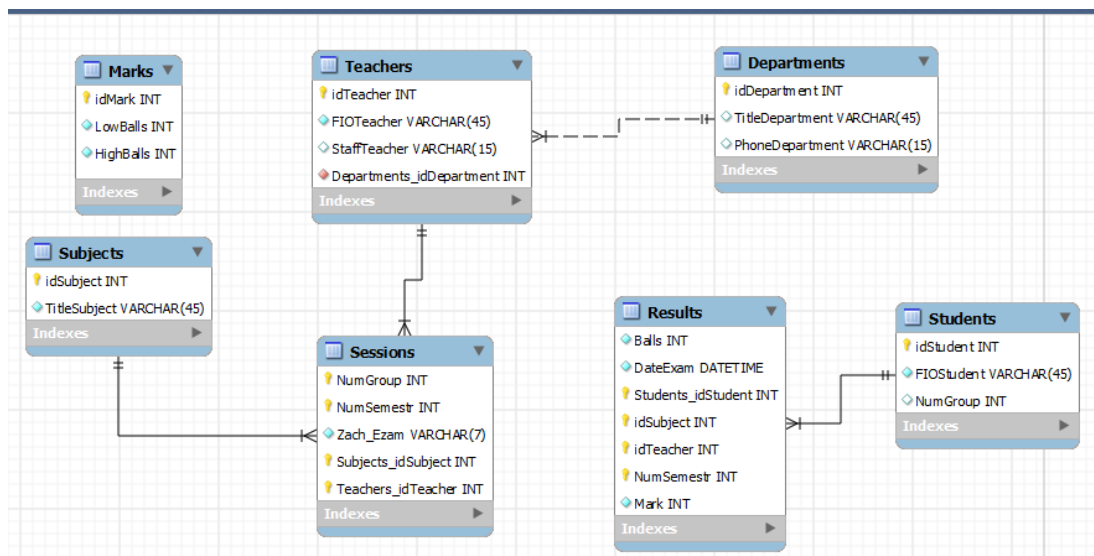


Рис. 11. Модель таблиц базы данных «Деканат» с указанием связей.

Отметим некоторую избыточность таблицы результатов относительно номера группы. Требуется обеспечить, чтобы номер группы и студенты были согласованы по таблицам студентов и результатов сессии.

### ***Построение модели в оболочке dbForge Studio для SQL Server***

Новую модель (диаграмму) базы данных можно создать с помощью меню «База данных»-> «Диаграмма БД».

Окно редактирования новой диаграммы состоит из полотна, на которое можно наносить новые таблицы с помощью визуальных средств редактирования, создать столбцы (атрибуты) таблиц и с помощью панели инструментов создать связи между таблицами.

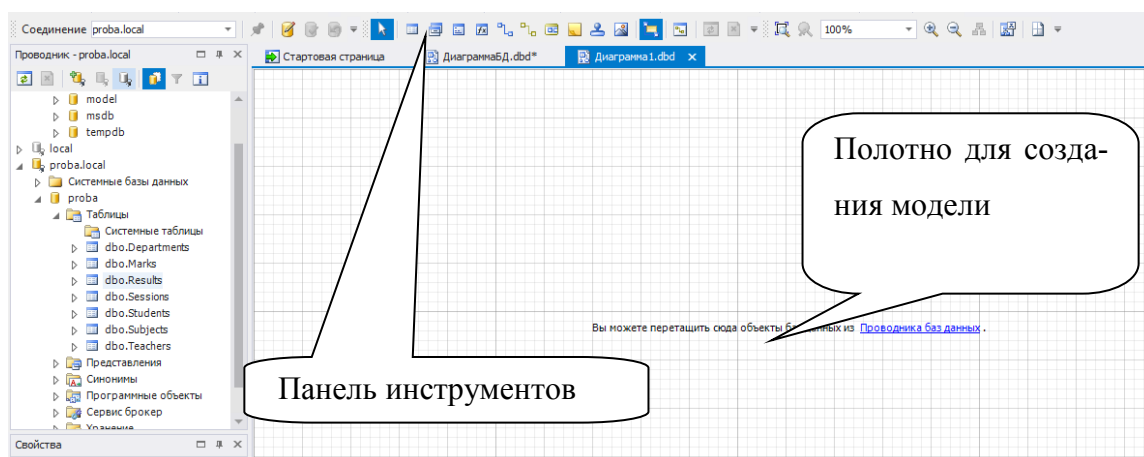


Рис.12. Окно редактирования диаграммы базы данных.

На панели инструментов следует отметить пока только две кнопки «Новая таблица», «Новая связь», которые позволяют создать новую таблицу, определив ее состав столбцов, первичные ключи и основные ограничения, и создать связи между таблицами, определив тем самым ограничения внешнего ключа.

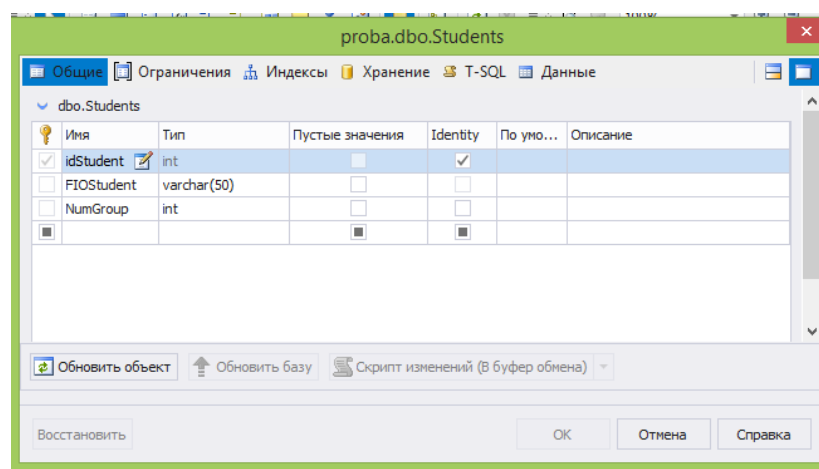


Рис. 13. Окно создания столбцов таблицы.

Для столбцов можно задать простые ограничения: допустимы ли пустые значения и определяет ли столбец поле-счетчик. Кроме того, можно выбрать столбцы, определяющие первичный ключ.

При создании связи требуется «нарисовать» мышью линию от дочерней таблицы к родительской. Для подтверждения параметров связи будет



показано окно, в котором нужно уточнить имена полей родительской и дочерней таблиц, которые будут связаны ограничением внешнего ключа:

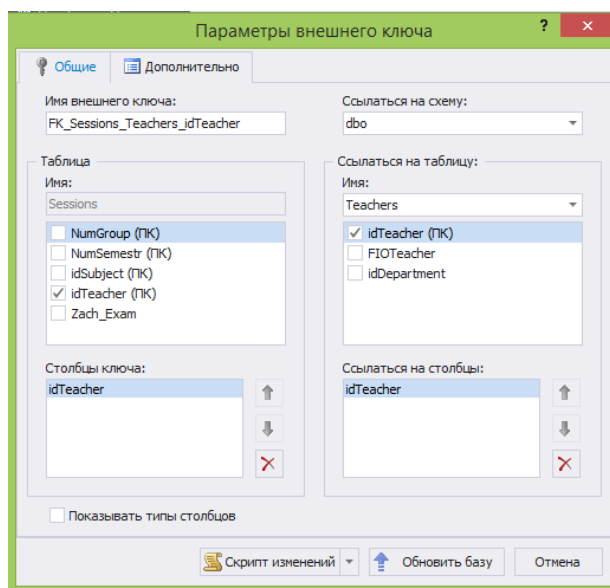


Рис. 14. Окно задания параметров внешнего ключа.

На вкладках «Ограничения» и «Индексы» можно увидеть все ограничения, которые сгенерируются в базе данных применительно к этой таблице. На вкладке T-SQL можно увидеть SQL-команду, выполнение которой эквивалентно выполнению всех сделанных настроек.

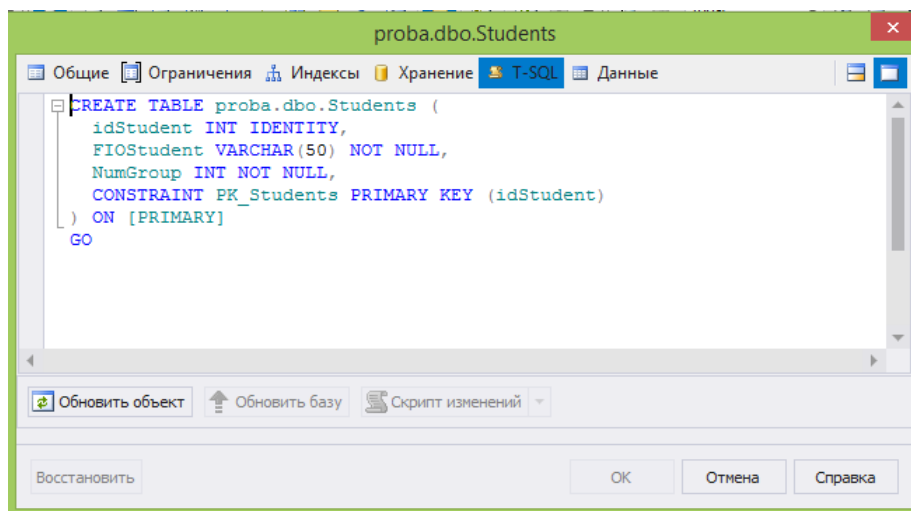


Рис. 15. Команда SQL создания таблицы «Студенты».

Отметим, что построитель модели синхронизирует все действия пользователя с базой данных, создавая указанные таблицы вместе со всеми ограничениями.

Таким образом, будет получена следующая модель:

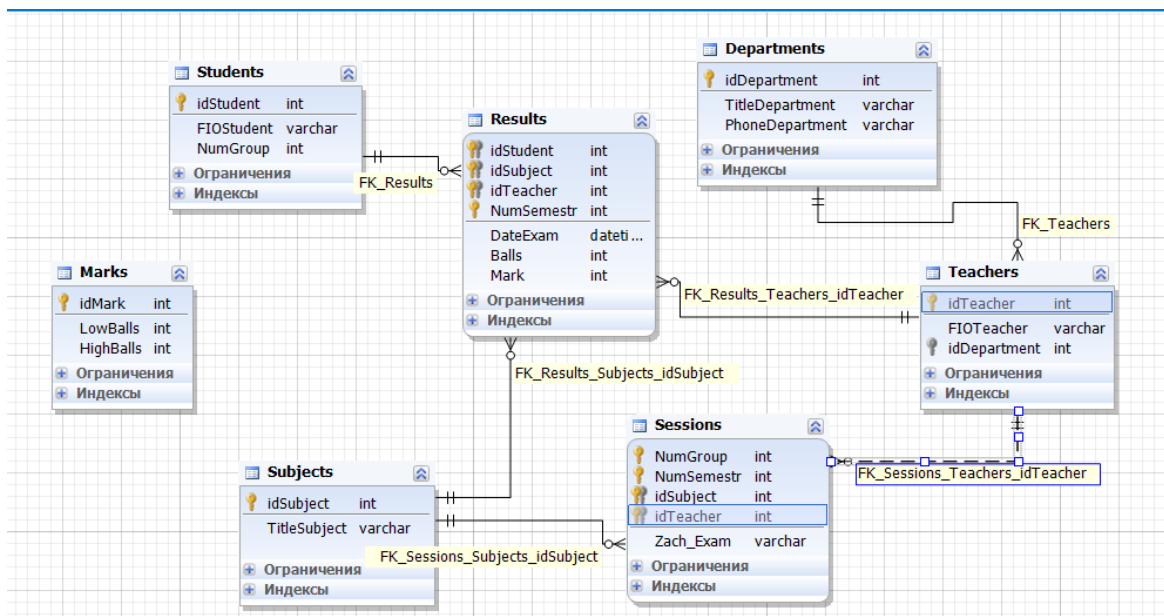


Рис. 16. Модель данных, построенная с помощью dbForge Studio.

Замечания относительно синхронизации номера группы в таблицах «Сессия» и «Студент» остаются на уровне модели нерешенным.

Аналогичным образом создается модель и, соответственно, база данных в среде dbForge Studio для MySQL.

Для PostgreSQL в стандартный набор инструмент формирования модели данных не входит. Поэтому состав таблиц нужно будет создать или с помощью специального SQL-оператора, или с помощью конструкторов таблиц:

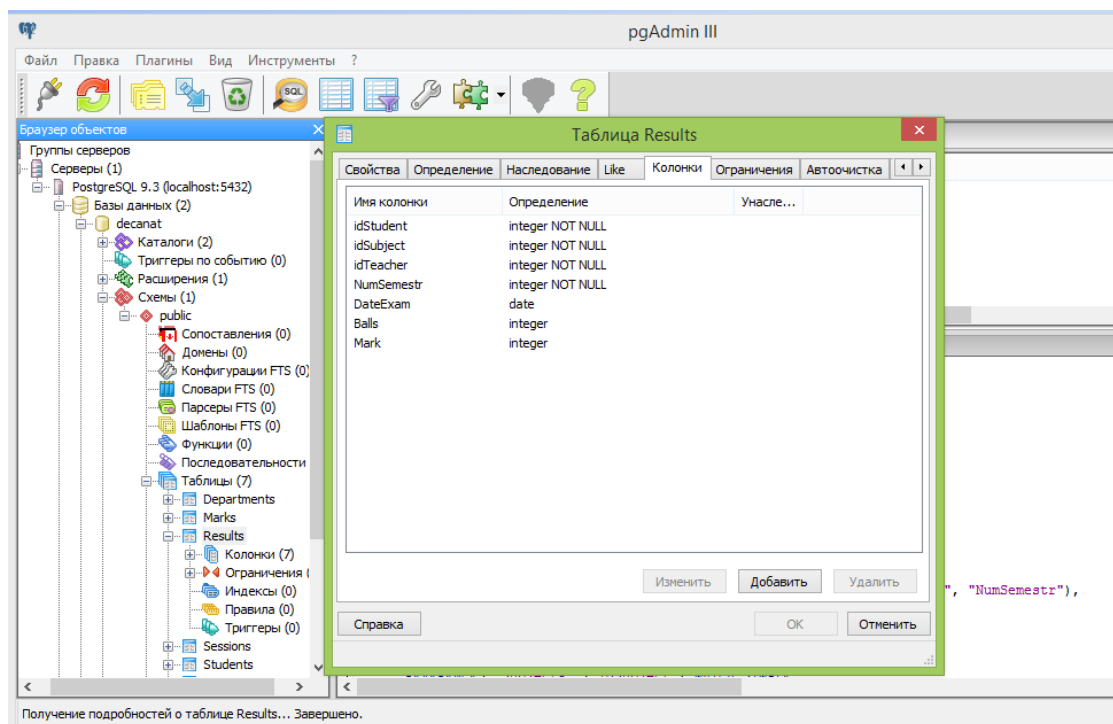


Рис. 17. Вид окна редактирования таблицы.

Действия с базой данных можно производить с помощью контекстного меню соответствующего элемента (таблицы, столбца, ограничения) в дереве объектов сервера. Настройки любого элемента производятся с помощью пункта контекстного меню «Свойства».

## 1.2. ПЕРЕНОС БАЗЫ ДАННЫХ НА ДРУГОЙ СЕРВЕР

Любое СУБД имеет средства резервного копирования базы данных. Такому копированию подвергаются как метаданные (структура данных базы), так и сами данные. Конечно, каждое СУБД имеет свои собственные форматы, но традиционным форматом является сохранение в виде последовательности SQL-команд (создания, вставки, изменения) (SQL-скрипт), выполнение которых приведет к текущему состоянию базы данных.

В оболочке dbForge Studio для SQL Server создание резервной копии (backup) можно осуществить двумя способами:

1. пункт меню «База данных» -> «Задачи» -> «Резервное копирование» (соответственно, для восстановления из резервной копии используется пункт меню «База данных» -> «Задачи» -> «Восстановление»). Этот способ связан с использованием специального формата MS SQL Server.

2. Генерация SQL-скрипта осуществляется с помощью пункта меню «База данных» -> «Задачи» -> «Сгенерировать скрипт...».

Кстати, многие важные опции, доступные через меню, доступны и на стартовой странице приложения, чтобы можно было получить к ним быстрый доступ:

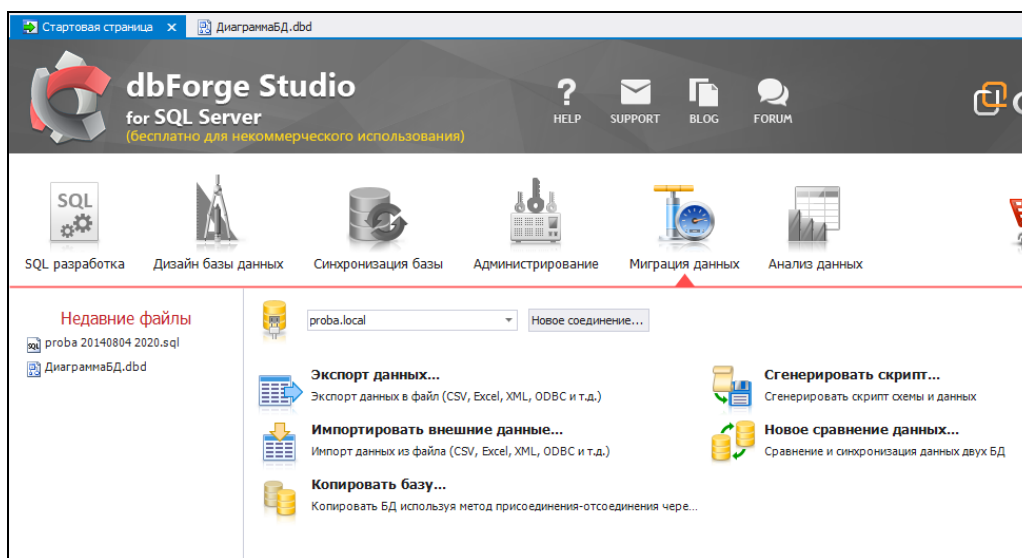


Рис. 18. Вид стартовой страницы для вкладки «Миграция данных».

В результате будет сгенерирован файл, содержащий следующие SQL-команды. Выделим полужирным шрифтом те команды, которые касаются создания базы данных и всех ее таблиц, а также определение ограничений:

```
--
-- Скрипт сгенерирован Devart dbForge Studio for SQL Server, Версия 3.8.180.1
-- Домашняя страница продукта: http://www.devart.com/ru/dbforge/sql/studio
-- Дата скрипта: 04.08.2014 23:36:06
-- Версия сервера: 11.00.2100
-- Версия клиента:
--
```

```
USE master
GO
```

```

IF DB_NAME() <> N'master' SET NOEXEC ON

--
-- Создать базу данных "proba"
--
PRINT (N'Создать базу данных "proba"')
GO

CREATE DATABASE proba
ON PRIMARY(
    NAME = N'proba',
    FILENAME = N'c:\Program Files\Microsoft SQL Serv-
er\MSSQL11.SQLEXPRESS\MSSQL\DATA\proba.mdf',
    SIZE = 4160KB,
    MAXSIZE = UNLIMITED,
    FILEGROWTH = 1024KB
)
LOG ON(
    NAME = N'proba_log',
    FILENAME = N'c:\Program Files\Microsoft SQL Serv-
er\MSSQL11.SQLEXPRESS\MSSQL\DATA\proba_log.ldf',
    SIZE = 1040KB,
    MAXSIZE = UNLIMITED,
    FILEGROWTH = 10%
)
GO

--
-- Изменить базу данных
--
PRINT (N'Изменить базу данных')
GO
ALTER DATABASE proba
SET
    ANSI_NULL_DEFAULT OFF,
    ANSI_NULLS OFF,
    ANSI_PADDING OFF,
    ANSI_WARNINGS OFF,
    ARITHABORT OFF,
    AUTO_CLOSE ON,
    AUTO_CREATE_STATISTICS ON,
    AUTO_SHRINK OFF,
    AUTO_UPDATE_STATISTICS ON,
    AUTO_UPDATE_STATISTICS_ASYNC OFF,
    COMPATIBILITY_LEVEL = 110,
    CONCAT_NULL_YIELDS_NULL OFF,
    CONTAINMENT = NONE,
    CURSOR_CLOSE_ON_COMMIT OFF,
    CURSOR_DEFAULT GLOBAL,
    DATE_CORRELATION_OPTIMIZATION OFF,
    DB_CHAINING OFF,
    HONOR_BROKER_PRIORITY OFF,
    MULTI_USER,
    NUMERIC_ROUNDABORT OFF,
    PAGE_VERIFY CHECKSUM,
    PARAMETERIZATION SIMPLE,

```

```

    QUOTED_IDENTIFIER OFF,
    READ_COMMITTED_SNAPSHOT OFF,
    RECOVERY SIMPLE,
    RECURSIVE_TRIGGERS OFF,
    TRUSTWORTHY OFF
    WITH ROLLBACK IMMEDIATE
GO

ALTER DATABASE proba
    SET ENABLE_BROKER
GO

ALTER DATABASE proba
    SET ALLOW_SNAPSHOT_ISOLATION OFF
GO

ALTER DATABASE proba
    SET FILESTREAM (NON_TRANSACTED_ACCESS = OFF)
GO
USE proba
GO
IF DB_NAME() <> N'proba' SET NOEXEC ON
GO

--
-- Создать таблицу "dbo.Teachers"
--
PRINT (N'Создать таблицу "dbo.Teachers"')
GO
CREATE TABLE dbo.Teachers (
    idTeacher int IDENTITY,
    FIOTeacher varchar(50) NOT NULL,
    idDepartment int NOT NULL,
    CONSTRAINT PK_Teachers PRIMARY KEY (idTeacher)
)
ON [PRIMARY]
GO

--
-- Создать таблицу "dbo.Subjects"
--
PRINT (N'Создать таблицу "dbo.Subjects"')
GO
CREATE TABLE dbo.Subjects (
    idSubject int IDENTITY,
    TitleSubject varchar(50) NOT NULL,
    CONSTRAINT PK_Subjects PRIMARY KEY (idSubject)
)
ON [PRIMARY]
GO

--
-- Создать таблицу "dbo.Students"
--
PRINT (N'Создать таблицу "dbo.Students"')
GO

```

```

CREATE TABLE dbo.Students (
    idStudent int IDENTITY,
    FIOStudent varchar(50) NOT NULL,
    NumGroup int NOT NULL,
    CONSTRAINT PK_Students PRIMARY KEY (idStudent)
)
ON [PRIMARY]
GO

--
-- Создать таблицу "dbo.Sessions"
--
PRINT (N'Создать таблицу "dbo.Sessions"')
GO
CREATE TABLE dbo.Sessions (
    NumGroup int NOT NULL,
    NumSemestr int NOT NULL,
    idSubject int NOT NULL,
    idTeacher int NOT NULL,
    Zach_Exam varchar(7) NOT NULL,
    CONSTRAINT PK_Sessions PRIMARY KEY (NumGroup, NumSemestr, idSubject, idTeacher)
)
ON [PRIMARY]
GO

--
-- Создать таблицу "dbo.Departments"
--
PRINT (N'Создать таблицу "dbo.Departments"')
GO
CREATE TABLE dbo.Departments (
    idDepartment int IDENTITY,
    TitleDepartment varchar(50) NOT NULL,
    PhoneDepartment varchar(50) NOT NULL,
    CONSTRAINT PK_Departments PRIMARY KEY (idDepartment)
)
ON [PRIMARY]
GO

--
-- Создать таблицу "dbo.Results"
--
PRINT (N'Создать таблицу "dbo.Results"')
GO
CREATE TABLE dbo.Results (
    idStudent int NOT NULL,
    idSubject int NOT NULL,
    idTeacher int NOT NULL,
    DateExam datetime NOT NULL,
    Balls int NOT NULL,
    Mark int NOT NULL,
    NumSemestr int NOT NULL,
    CONSTRAINT PK_Results PRIMARY KEY (idStudent, idSubject, idTeacher, NumSemestr)
)

```

```

ON [PRIMARY]
GO

--
-- Создать таблицу "dbo.Marks"
--
PRINT (N'Создать таблицу "dbo.Marks"')
GO
CREATE TABLE dbo.Marks (
    idMark int IDENTITY,
    LowBalls int NOT NULL,
    HighBalls int NOT NULL,
    CONSTRAINT PK_Marks PRIMARY KEY (idMark)
)
ON [PRIMARY]
GO
--
-- секция для команд вставки данных из всех таблиц - ее пропустим
--
-- Создать внешний ключ "FK_Teachers" для объекта типа таблица "dbo.Teachers"
--
PRINT (N'Создать внешний ключ "FK_Teachers" для объекта типа таблица
"dbo.Teachers"')
GO
ALTER TABLE dbo.Teachers
    ADD CONSTRAINT FK_Teachers FOREIGN KEY (idDepartment) REFERENCES
dbo.Departments (idDepartment)
GO
--
-- Создать внешний ключ "FK_Sessions_Subjects_idSubject" для объекта типа табли-
ца "dbo.Sessions"
--
PRINT (N'Создать внешний ключ "FK_Sessions_Subjects_idSubject" для объекта типа
таблица "dbo.Sessions"')
GO
ALTER TABLE dbo.Sessions
    ADD CONSTRAINT FK_Sessions_Subjects_idSubject FOREIGN KEY (idSubject) REFER-
ENCES dbo.Subjects (idSubject)
GO
--
-- Создать внешний ключ "FK_Sessions_Teachers_idTeacher" для объекта типа табли-
ца "dbo.Sessions"
--
PRINT (N'Создать внешний ключ "FK_Sessions_Teachers_idTeacher" для объекта типа
таблица "dbo.Sessions"')
GO
ALTER TABLE dbo.Sessions
    ADD CONSTRAINT FK_Sessions_Teachers_idTeacher FOREIGN KEY (idTeacher) REFER-
ENCES dbo.Teachers (idTeacher)
GO
--
-- Создать внешний ключ "FK_Results" для объекта типа таблица "dbo.Results"
--

```



```

PRINT (N'Создать внешний ключ "FK_Results" для объекта типа таблица
"dbo.Results"')
GO
ALTER TABLE dbo.Results
    ADD CONSTRAINT FK_Results FOREIGN KEY (idStudent) REFERENCES dbo.Students
(idStudent)
GO

--
-- Создать внешний ключ "FK_Results_Subjects_idSubject" для объекта типа таблица
"dbo.Results"
--
PRINT (N'Создать внешний ключ "FK_Results_Subjects_idSubject" для объекта типа
таблица "dbo.Results"')
GO
ALTER TABLE dbo.Results
    ADD CONSTRAINT FK_Results_Subjects_idSubject FOREIGN KEY (idSubject) REFER-
ENCES dbo.Subjects (idSubject)
GO

--
-- Создать внешний ключ "FK_Results_Teachers_idTeacher" для объекта типа таблица
"dbo.Results"
--
PRINT (N'Создать внешний ключ "FK_Results_Teachers_idTeacher" для объекта типа
таблица "dbo.Results"')
GO
ALTER TABLE dbo.Results
    ADD CONSTRAINT FK_Results_Teachers_idTeacher FOREIGN KEY (idTeacher) REFER-
ENCES dbo.Teachers (idTeacher)
GO
SET NOEXEC OFF
GO

```

При работе с оболочкой dbForge Studio для MySQL используется другой путь к пункту меню: «База данных» -> «Резервная копия» -> «Создать резервную копию БД» (аналогично можно использовать гиперссылку на стартовой странице в разделе «Миграция данных»). Приведем содержимое этого файла:

```

--
-- Скрипт сгенерирован Devart dbForge Studio for MySQL, Версия 6.2.233.0
-- Домашняя страница продукта: http://www.devart.com/ru/dbforge/mysql/studio
-- Дата скрипта: 04.08.2014 23:47:11
-- Версия сервера: 5.0.67-community-nt
-- Версия клиента: 4.1
--
--
-- Отключение внешних ключей
--
/*!40014 SET @OLD_FOREIGN_KEY_CHECKS=@@FOREIGN_KEY_CHECKS, FOREIGN_KEY_CHECKS=0
*/;
--

```

```

-- Установить режим SQL (SQL mode)
--
/*!40101 SET @OLD_SQL_MODE=@@SQL_MODE, SQL_MODE='NO_AUTO_VALUE_ON_ZERO' */;
--
-- Установка базы данных по умолчанию
--
USE decanat;
--
-- Описание для таблицы departments
--
DROP TABLE IF EXISTS departments;
CREATE TABLE departments (
  idDepartment INT(11) NOT NULL AUTO_INCREMENT,
  TitleDepartment VARCHAR(255) NOT NULL,
  PhoneDepartment VARCHAR(255) NOT NULL,
  PRIMARY KEY (idDepartment)
)
ENGINE = INNODB
AUTO_INCREMENT = 1
CHARACTER SET utf8
COLLATE utf8_general_ci;

--
-- Описание для таблицы marks
--
DROP TABLE IF EXISTS marks;
CREATE TABLE marks (
  idMark INT(11) NOT NULL,
  LowBalls INT(11) NOT NULL,
  HighBalls INT(11) NOT NULL,
  PRIMARY KEY (idMark)
)
ENGINE = INNODB
CHARACTER SET utf8
COLLATE utf8_general_ci;

--
-- Описание для таблицы students
--
DROP TABLE IF EXISTS students;
CREATE TABLE students (
  idStudent INT(11) NOT NULL AUTO_INCREMENT,
  FIOStudent VARCHAR(255) NOT NULL,
  NumGroup INT(11) NOT NULL,
  PRIMARY KEY (idStudent)
)
ENGINE = INNODB
AUTO_INCREMENT = 1
CHARACTER SET utf8
COLLATE utf8_general_ci;

--
-- Описание для таблицы subjects
--
DROP TABLE IF EXISTS subjects;
CREATE TABLE subjects (

```

```

    idSubject INT(11) NOT NULL AUTO_INCREMENT,
    TitleSubject VARCHAR(255) NOT NULL,
    PRIMARY KEY (idSubject)
)
ENGINE = INNODB
AUTO_INCREMENT = 1
CHARACTER SET utf8
COLLATE utf8_general_ci;

--
-- Описание для таблицы teachers
--
DROP TABLE IF EXISTS teachers;
CREATE TABLE teachers (
    idTeacher INT(11) NOT NULL AUTO_INCREMENT,
    FIOTeacher VARCHAR(255) NOT NULL,
    idDepartment INT(11) NOT NULL,
    PRIMARY KEY (idTeacher),
    CONSTRAINT FK_teachers_departments_idDepartment FOREIGN KEY (idDepartment)
        REFERENCES departments(idDepartment) ON DELETE NO ACTION ON UPDATE NO ACTION
)
ENGINE = INNODB
AUTO_INCREMENT = 1
CHARACTER SET utf8
COLLATE utf8_general_ci;

--
-- Описание для таблицы results
--
DROP TABLE IF EXISTS results;
CREATE TABLE results (
    idStudent INT(11) NOT NULL,
    idSubject INT(11) NOT NULL,
    idTeacher INT(11) NOT NULL,
    DateExam DATETIME NOT NULL,
    NumSemestr INT(11) NOT NULL,
    Balls INT(11) NOT NULL,
    Mark INT(11) NOT NULL,
    PRIMARY KEY (idStudent, idSubject, idTeacher, NumSemestr),
    CONSTRAINT FK_results_students_idStudent FOREIGN KEY (idStudent)
        REFERENCES students(idStudent) ON DELETE NO ACTION ON UPDATE NO ACTION,
    CONSTRAINT FK_results_subjects_idSubject FOREIGN KEY (idSubject)
        REFERENCES subjects(idSubject) ON DELETE NO ACTION ON UPDATE NO ACTION,
    CONSTRAINT FK_results_teachers_idTeacher FOREIGN KEY (idTeacher)
        REFERENCES teachers(idTeacher) ON DELETE NO ACTION ON UPDATE NO ACTION
)
ENGINE = INNODB
CHARACTER SET utf8
COLLATE utf8_general_ci;

--
-- Описание для таблицы sessions
--
DROP TABLE IF EXISTS sessions;
CREATE TABLE sessions (
    NumGroup INT(11) NOT NULL,

```

```

NumSemestr INT(11) NOT NULL,
idSubject INT(11) NOT NULL,
idTeacher INT(11) NOT NULL,
Zach_Exam VARCHAR(7) NOT NULL,
PRIMARY KEY (NumGroup, NumSemestr, idSubject, idTeacher),
CONSTRAINT FK_sessions_subjects_idSubject FOREIGN KEY (idSubject)
    REFERENCES subjects(idSubject) ON DELETE NO ACTION ON UPDATE NO ACTION,
CONSTRAINT FK_sessions_teachers_idTeacher FOREIGN KEY (idTeacher)
    REFERENCES teachers(idTeacher) ON DELETE NO ACTION ON UPDATE NO ACTION
)
ENGINE = INNODB
CHARACTER SET utf8
COLLATE utf8_general_ci;

--
-- секция для команд вставки данных из таблиц
--
--
-- Включение внешних ключей
--
/*!40014 SET FOREIGN_KEY_CHECKS=@OLD_FOREIGN_KEY_CHECKS */;

```

В PostgreSQL резервную копию можно создать с помощью пункта контекстного меню «Резервная копия» для элемента дерева, соответствующего копируемой базе данных. В результате будет сгенерирован следующий SQL-код:

```

--
-- PostgreSQL database dump
--
-- Dumped from database version 9.3.5
-- Dumped by pg_dump version 9.3.5
-- Started on 2014-08-04 23:54:32

SET statement_timeout = 0;
SET lock_timeout = 0;
SET client_encoding = 'UTF8';
SET standard_conforming_strings = on;
SET check_function_bodies = false;
SET client_min_messages = warning;
--
-- TOC entry 177 (class 3079 OID 11750)
-- Name: plpgsql; Type: EXTENSION; Schema: -; Owner:
--

CREATE EXTENSION IF NOT EXISTS plpgsql WITH SCHEMA pg_catalog;

--
-- TOC entry 1991 (class 0 OID 0)
-- Dependencies: 177
-- Name: EXTENSION plpgsql; Type: COMMENT; Schema: -; Owner:
--

```

```

COMMENT ON EXTENSION plpgsql IS 'PL/pgSQL procedural language';

SET search_path = public, pg_catalog;
SET default_tablespace = '';
SET default_with_oids = false;
--
-- TOC entry 171 (class 1259 OID 16397)
-- Name: Departments; Type: TABLE; Schema: public; Owner: postgres; Tablespace:
--

CREATE TABLE "Departments" (
    "idDepartment" oid NOT NULL,
    "TitleDepartment" text NOT NULL,
    "PhoneDepartment" text
);

ALTER TABLE public."Departments" OWNER TO postgres;

--
-- TOC entry 170 (class 1259 OID 16394)
-- Name: Marks; Type: TABLE; Schema: public; Owner: postgres; Tablespace:
--

CREATE TABLE "Marks" (
    "idMark" integer NOT NULL,
    "LowBalls" integer NOT NULL,
    "HighBalls" integer NOT NULL
);

ALTER TABLE public."Marks" OWNER TO postgres;

--
-- TOC entry 176 (class 1259 OID 16431)
-- Name: Results; Type: TABLE; Schema: public; Owner: postgres; Tablespace:
--

CREATE TABLE "Results" (
    "idStudent" integer NOT NULL,
    "idSubject" integer NOT NULL,
    "idTeacher" integer NOT NULL,
    "NumSemestr" integer NOT NULL,
    "DateExam" date,
    "Balls" integer,
    "Mark" integer
);

ALTER TABLE public."Results" OWNER TO postgres;

--
-- TOC entry 174 (class 1259 OID 16415)
-- Name: Sessions; Type: TABLE; Schema: public; Owner: postgres; Tablespace:
--

CREATE TABLE "Sessions" (
    "NumGroup" integer NOT NULL,
    "NumSemestr" integer NOT NULL,
    "idSubject" integer NOT NULL,
    "idTeacher" integer NOT NULL,
    "Zach_Exam" text

```

```

);

ALTER TABLE public."Sessions" OWNER TO postgres;
--
-- TOC entry 175 (class 1259 OID 16423)
-- Name: Students; Type: TABLE; Schema: public; Owner: postgres; Tablespace:
--

CREATE TABLE "Students" (
    "idStudent" oid NOT NULL,
    "FIOStudent" text,
    "NumGroup" integer
);

ALTER TABLE public."Students" OWNER TO postgres;
--
-- TOC entry 172 (class 1259 OID 16403)
-- Name: Subjects; Type: TABLE; Schema: public; Owner: postgres; Tablespace:
--

CREATE TABLE "Subjects" (
    "idSubject" oid NOT NULL,
    "TitleSubject" text NOT NULL
);

ALTER TABLE public."Subjects" OWNER TO postgres;
--
-- TOC entry 173 (class 1259 OID 16409)
-- Name: Teachers; Type: TABLE; Schema: public; Owner: postgres; Tablespace:
--

CREATE TABLE "Teachers" (
    "idTeacher" oid NOT NULL,
    "FIOTeacher" text NOT NULL,
    "idDepartment" integer NOT NULL
);

ALTER TABLE public."Teachers" OWNER TO postgres;
--
-- TOC entry 1978 (class 0 OID 16397)
-- Dependencies: 171
-- Data for Name: Departments; Type: TABLE DATA; Schema: public; Owner: postgres
--

COPY "Departments" ("idDepartment", "TitleDepartment", "PhoneDepartment") FROM
stdin;
\

--
-- TOC entry 1977 (class 0 OID 16394)
-- Dependencies: 170
-- Data for Name: Marks; Type: TABLE DATA; Schema: public; Owner: postgres
--

COPY "Marks" ("idMark", "LowBalls", "HighBalls") FROM stdin;

```

```

\..

--
-- TOC entry 1983 (class 0 OID 16431)
-- Dependencies: 176
-- Data for Name: Results; Type: TABLE DATA; Schema: public; Owner: postgres
--

COPY "Results" ("idStudent", "idSubject", "idTeacher", "NumSemestr", "DateExam",
"Balls", "Mark") FROM stdin;
\..

--
-- TOC entry 1981 (class 0 OID 16415)
-- Dependencies: 174
-- Data for Name: Sessions; Type: TABLE DATA; Schema: public; Owner: postgres
--

COPY "Sessions" ("NumGroup", "NumSemestr", "idSubject", "idTeacher",
"Zach_Exam") FROM stdin;
\..

--
-- TOC entry 1982 (class 0 OID 16423)
-- Dependencies: 175
-- Data for Name: Students; Type: TABLE DATA; Schema: public; Owner: postgres
--

COPY "Students" ("idStudent", "FIOSStudent", "NumGroup") FROM stdin;
\..

--
-- TOC entry 1979 (class 0 OID 16403)
-- Dependencies: 172
-- Data for Name: Subjects; Type: TABLE DATA; Schema: public; Owner: postgres
--

COPY "Subjects" ("idSubject", "TitleSubject") FROM stdin;
\..

--
-- TOC entry 1980 (class 0 OID 16409)
-- Dependencies: 173
-- Data for Name: Teachers; Type: TABLE DATA; Schema: public; Owner: postgres
--

COPY "Teachers" ("idTeacher", "FIOTeacher", "idDepartment") FROM stdin;
\..

--
-- TOC entry 1853 (class 2606 OID 16439)
-- Name: pk_department; Type: CONSTRAINT; Schema: public; Owner: postgres; Ta-
blespace:
--

ALTER TABLE ONLY "Departments"

```

```

    ADD CONSTRAINT pk_department PRIMARY KEY ("idDepartment");

--
-- TOC entry 1851 (class 2606 OID 16441)
-- Name: pk_mark; Type: CONSTRAINT; Schema: public; Owner: postgres; Tablespace:
--

ALTER TABLE ONLY "Marks"
    ADD CONSTRAINT pk_mark PRIMARY KEY ("idMark");

--
-- TOC entry 1863 (class 2606 OID 16435)
-- Name: pk_results; Type: CONSTRAINT; Schema: public; Owner: postgres; Table-
space:
--

ALTER TABLE ONLY "Results"
    ADD CONSTRAINT pk_results PRIMARY KEY ("idStudent", "idTeacher", "idSub-
ject", "NumSemestr");

--
-- TOC entry 1859 (class 2606 OID 16422)
-- Name: pk_sessions; Type: CONSTRAINT; Schema: public; Owner: postgres; Table-
space:
--

ALTER TABLE ONLY "Sessions"
    ADD CONSTRAINT pk_sessions PRIMARY KEY ("NumGroup", "NumSemestr", "idSub-
ject", "idTeacher");

--
-- TOC entry 1861 (class 2606 OID 16430)
-- Name: pk_students; Type: CONSTRAINT; Schema: public; Owner: postgres; Table-
space:
--

ALTER TABLE ONLY "Students"
    ADD CONSTRAINT pk_students PRIMARY KEY ("idStudent");

--
-- TOC entry 1855 (class 2606 OID 16443)
-- Name: pk_subject; Type: CONSTRAINT; Schema: public; Owner: postgres; Table-
space:
--

ALTER TABLE ONLY "Subjects"
    ADD CONSTRAINT pk_subject PRIMARY KEY ("idSubject");

--
-- TOC entry 1857 (class 2606 OID 16445)
-- Name: pk_teacher; Type: CONSTRAINT; Schema: public; Owner: postgres; Table-
space:
--

ALTER TABLE ONLY "Teachers"
    ADD CONSTRAINT pk_teacher PRIMARY KEY ("idTeacher");

```



```

--
-- TOC entry 1864 (class 2606 OID 16446)
-- Name: fk_dep_tea; Type: FK CONSTRAINT; Schema: public; Owner: postgres
--

ALTER TABLE ONLY "Teachers"
    ADD CONSTRAINT fk_dep_tea FOREIGN KEY ("idDepartment") REFERENCES "Depart-
ments"("idDepartment");

--
-- TOC entry 1867 (class 2606 OID 16461)
-- Name: fk_res_stud; Type: FK CONSTRAINT; Schema: public; Owner: postgres
--

ALTER TABLE ONLY "Results"
    ADD CONSTRAINT fk_res_stud FOREIGN KEY ("idStudent") REFERENCES "Stu-
dents"("idStudent");

--
-- TOC entry 1868 (class 2606 OID 16466)
-- Name: fk_res_sub; Type: FK CONSTRAINT; Schema: public; Owner: postgres
--

ALTER TABLE ONLY "Results"
    ADD CONSTRAINT fk_res_sub FOREIGN KEY ("idSubject") REFERENCES "Sub-
jects"("idSubject");

--
-- TOC entry 1869 (class 2606 OID 16471)
-- Name: fk_res_tea; Type: FK CONSTRAINT; Schema: public; Owner: postgres
--

ALTER TABLE ONLY "Results"
    ADD CONSTRAINT fk_res_tea FOREIGN KEY ("idTeacher") REFERENCES "Teach-
ers"("idTeacher");

--
-- TOC entry 1865 (class 2606 OID 16451)
-- Name: fk_sess_subj; Type: FK CONSTRAINT; Schema: public; Owner: postgres
--

ALTER TABLE ONLY "Sessions"
    ADD CONSTRAINT fk_sess_subj FOREIGN KEY ("idSubject") REFERENCES "Sub-
jects"("idSubject");

--
-- TOC entry 1866 (class 2606 OID 16456)
-- Name: fk_sess_tea; Type: FK CONSTRAINT; Schema: public; Owner: postgres
--

ALTER TABLE ONLY "Sessions"
    ADD CONSTRAINT fk_sess_tea FOREIGN KEY ("idTeacher") REFERENCES "Teach-
ers"("idTeacher");

```

```
--
-- TOC entry 1990 (class 0 OID 0)
-- Dependencies: 5
-- Name: public; Type: ACL; Schema: -; Owner: postgres
--


REVOKE ALL ON SCHEMA public FROM PUBLIC;
REVOKE ALL ON SCHEMA public FROM postgres;
GRANT ALL ON SCHEMA public TO postgres;
GRANT ALL ON SCHEMA public TO PUBLIC;

-- Completed on 2014-08-04 23:54:32
--
-- PostgreSQL database dump complete
--
```

При внимательном рассмотрении трех сгенерированных скриптов видно, что основные команды по созданию таблиц и ограничений первичного и внешнего ключей почти не отличаются для этих трех СУБД. Дело в том, что язык SQL является стандартом для работы с базами данных и все современные реляционные базы данных стараются соблюдать этот стандарт. В основном, существенные различия заключаются только в используемых типах данных и небольших дополнениях, связанных с особенностями задания владельца таблицы (пользователя, который создал таблицу и, следовательно, имеет максимальные права для работы с ней), используемой кодировки и других настроек СУБД.

Нередко первоначальное проектирование выполняется с ошибками или недочетами (не все условия учтены, требуются новые столбцы или, напротив, какие-то столбцы являются лишними). Очевидно, что необходимы средства для обеспечения простого внесения изменений в таблицы. Этим средством является команда SQL **ALTER TABLE**, которую используют для корректировки списка столбцов таблицы и наложения разных ограничений как на отдельные столбцы, так и на таблицу в целом. Покажем на нескольких примерах, как можно использовать эту команду.

Выполнение SQL-команд осуществляется в оболочках с помощью специальных окон редактирования и выполнения SQL-скриптов. В dbForge Studio его можно создать с помощью меню «Новый»-> «SQL». В pgAdmin окно выполнения пользовательских запросов можно вызвать с помощью специальной кнопки

на панели инструментов . После создания ограничения можно увидеть как объекты соответствующих таблиц в дереве элементов базы данных (ограничения или индексы):

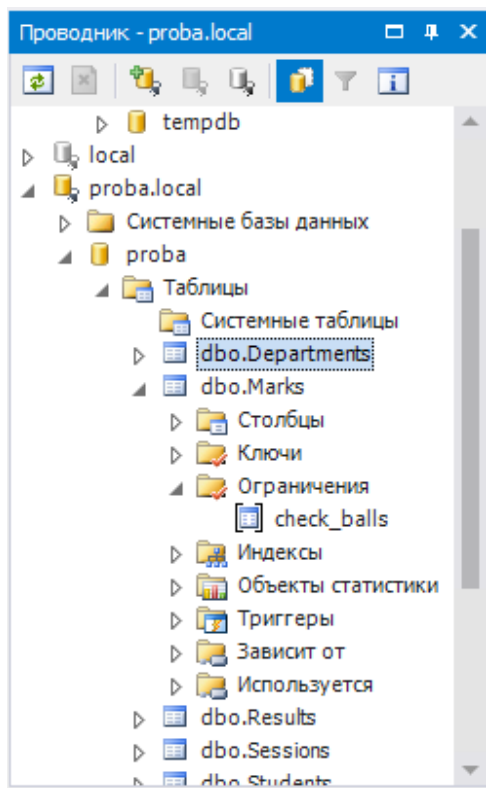


Рис. 19. Ограничение на проверку баллов в dbForge для SQL Server.

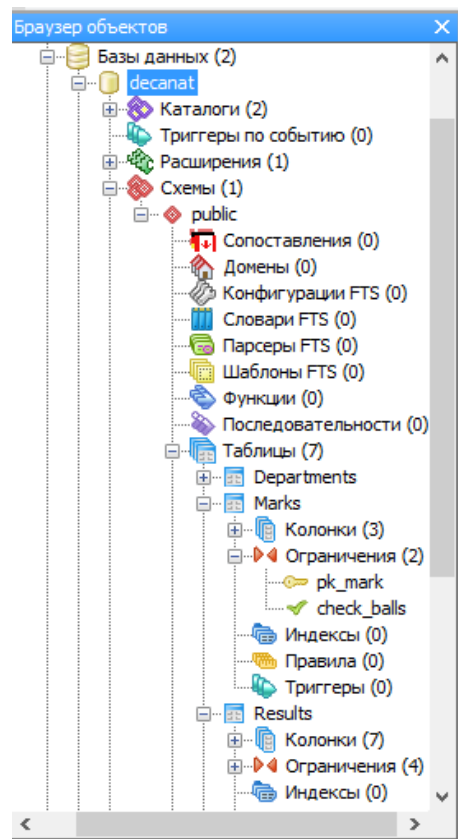


Рис.20. Ограничение на проверку баллов в pgAdmin.

Пример 1. Добавим ограничение уникальности на название кафедры в таблице Departments:

MS SQL Server, MySQL:

```
ALTER TABLE Departments ADD CONSTRAINT un_title
UNIQUE (TitleDepartment);
```

PostgreSQL (отличием является заключением в кавычки имен таблиц, ограничения, столбцов):

```
ALTER TABLE "Departments" ADD CONSTRAINT "un_title"
UNIQUE ("TitleDepartment");
```

Пример 2. Требуется добавить ограничение проверки условия для таблицы оценок, означающее, что нижняя граница баллов должна быть меньше верхней.

MS SQL Server:

```
ALTER TABLE Marks ADD CONSTRAINT check_balls  
CHECK (LowBalls<HighBalls);
```

Для MySQL эта команда выполняется, но как таковой объект базы данных не создается.

PostgreSQL:

```
ALTER TABLE "Marks" ADD CONSTRAINT "check_balls"  
CHECK ("LowBalls"<"HighBalls");
```

Пример 3. Требуется добавить ограничение проверки условия для поля телефона кафедры – телефон должен состоять из 7 цифр и иметь формат «xxx-xx-xx». Можно также, как и в предыдущем примере, добавить ограничение CHECK. Однако мы для демонстрации возможностей команды ALTER TABLE сначала удалим столбец телефона кафедры, а потом добавим новый столбец с учетом ограничения. В телефоне первая цифра 2 или 5, остальные цифры могут быть любыми. Для этого используется конструкция языка LIKE, задающая шаблон записи телефона (для PostgreSQL это конструкция SIMILAR TO):

MS SQL Server, MySQL:

```
ALTER TABLE Departments DROP COLUMN PhoneDepartment;  
ALTER TABLE Departments ADD PhoneDepartment VARCHAR(9) CHECK  
(PhoneDepartment LIKE '[2,5][0-9][0-9]-[0-9][0-9]-[0-9][0-9]');
```

PostgreSQL:

```
ALTER TABLE "Departments" DROP COLUMN "PhoneDepartment";  
ALTER TABLE "Departments" ADD "PhoneDepartment" text CHECK  
("PhoneDepartment" SIMILAR TO  
'(2|5)[0-9][0-9]-[0-9][0-9]-[0-9][0-9]');
```

Пример 4. Введем ограничение на согласованность баллов и оценки в таблице результатов сессии. Будем полагать для простоты, что в таблицу заносятся только положительные результаты сдачи зачетов и экзаменов. Таким об-

разом, оценка должна быть только 3, 4 или 5. При этом должна быть учтена согласованность баллов исходя из шкалы принятой балльно-рейтинговой системы:

#### MS SQL Server, MySQL:

```
ALTER TABLE Results ADD CONSTRAINT ch_res_marks CHECK (Mark IN (3,4,5) AND ((Mark=3 AND Balls BETWEEN 55 AND 70) OR (Mark=4 AND Balls BETWEEN 71 AND 85) OR (Mark=5 AND Balls BETWEEN 86 AND 100)));
```

#### PostgreSQL:

```
ALTER TABLE "Results" ADD CONSTRAINT "ch_res_marks" CHECK ("Mark" IN (3,4,5) AND (("Mark"=3 AND "Balls" BETWEEN 55 AND 70) OR ("Mark"=4 AND "Balls" BETWEEN 71 AND 85) OR ("Mark"=5 AND "Balls" BETWEEN 86 AND 100)));
```

Заметим, что для аналогичных целей была предназначена таблица Marks. В дальнейшем мы будем использовать ее.

Пример 5. В MySQL иногда требуется явно указать кодировку данных таблиц. Это также можно сделать с помощью команды ALTER TABLE:

```
ALTER TABLE `Departments` CONVERT TO CHARACTER SET utf8;
```

### 1.3. КОМАНДЫ МОДИФИКАЦИИ ДАННЫХ (DML)

Оболочки проектирования содержат средства визуального внесения данных в таблицы. Например, в dbForge Studio это делается с помощью контекстного меню таблицы, пункта «Редактировать таблицу», вкладка «Данные»:

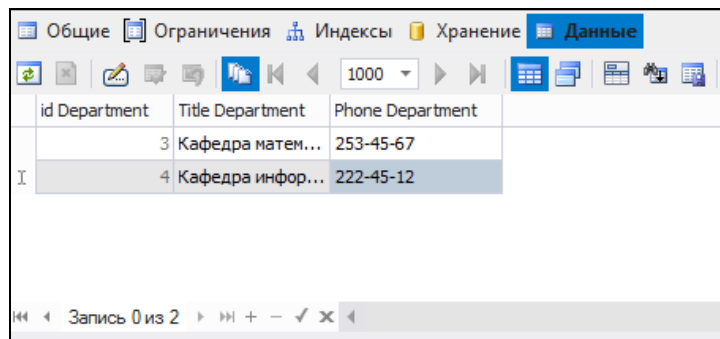


Рис. 21. Окно редактирования данных.

С помощью нижней панели можно добавлять, удалять записи и перемещаться по ним.

Сохранение данных происходит не сразу, а после закрытия окна. Но в процессе ввода данные проверяются на соответствие ограничениям. Например, внесем неправильный телефонный номер. В этом случае об ошибке будет сообщено следующим образом:

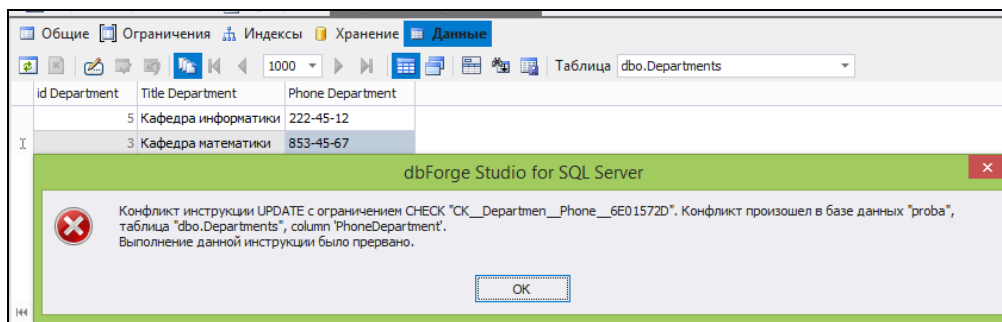


Рис. 22. Сообщение о нарушении ограничения.

Естественно, добавление записей (особенно массовое) может проводиться с помощью команд SQL, которые объединены в один сценарий. Итак, сценарий внесения данных в базу для MS SQL Server может быть таким:

```
USE proba
```

```
GO
```

```
-- вставка записей в таблицу Students
```

```
insert into Students (FIOSStudent,NumGroup)      VALUES      ('Иванов Иван  
Иванович',901);
```

```
-- и другие записи
```

```
GO
```

```
-- вставка записей в таблицу Departments
```

```
insert into Departments (TitleDepartment,PhoneDepartment)  VALUES  
('Кафедра математики','234-11-45');
```

```
-- и другие записи
```

```
GO
```

```
-- вставка записей в таблицу Teachers
```

```
insert into Teachers (FIOTeacher,idDepartment )  VALUES  ('Федосеев  
Александр Иванович', 6);
```

```
-- и другие записи
```

```
GO
```

```
-- вставка записей в таблицу Subjects
insert into Subjects (TitleSubject) VALUES ('Математический анализ');
-- и другие записи
GO

-- вставка записей в таблицу Sessions
insert into Sessions (NumGroup,NumSemestr, Zach_Exam,idSubject, idTeacher)
VALUES (901,1, 'зачет', 2,18);
-- и другие записи
GO

-- вставка записей в таблицу Marks
insert into Marks(idMark,LowBalls,HighBalls) VALUES (5, 86, 100);
-- и другие записи
GO
```

Отметим, что свойство счетчика в MS SQL Server для поля фиксирует все операции с таблицей, поэтому, к примеру, несмотря на отсутствие записей в таблице, новый номер может быть отличен от 1. Оператор **GO**, который используется в скрипте, специфичен для MS SQL Server и разделяет скрипт на неделимые блоки, которые выполняются полностью или не выполняются вообще. Удобнее всего выполнять этот сценарий блоками, чтобы между ними можно было бы убедиться в корректности использования значений в полях внешнего ключа.

Команды вставки для MySQL не отличаются от приведенного выше кода (за исключением команды GO). Также внимательно следует относиться к ключевым полям при установке связи внешнего ключа.

Сценарий для PostgreSQL имеет следующий вид. Отсутствие полей-счетчиков требует указывать ключевые поля во всех записях.

```
-- сценарий вставки записей в таблицы базы данных
insert into "Students" ("idStudent", "FIOStudent", "NumGroup")
VALUES (1, 'Иванов Иван Иванович', 901);
-- и другие записи
```

При нарушениях тех или иных ограничений оператор вставки не срабатывает. Так, повторная вставка записи или вставка записи с уже существующим первичным ключом будет запрещена. Например, повторно осуществляем вставку записи:

```
insert into Marks(idMark,LowBalls,HighBalls) VALUES (5, 86, 100);
```

команда выполнена не будет. Мы увидим сообщение об ошибке:

### MS SQL Server:

Ошибка: (53,1): Нарушено "PK\_Marks" ограничения PRIMARY KEY. Не удастся вставить повторяющийся ключ в объект "dbo.Marks". Повторяющееся значение ключа: (5).

### PostgreSQL:

ОШИБКА: повторяющееся значение ключа нарушает ограничение уникальности "pk\_mark"

SQL-состояние: 23505

Подробности: Ключ "("idMark")=(5)" уже существует.

### MySQL:

Duplicate entry '5' for key 1

При вставке записи с нарушением ограничения внешнего ключа также будет выведено сообщение об ошибке. Например, мы пытаемся вставить строку в таблицу сессии, в которой код преподавателя равен 100:

```
insert into Sessions (NumGroup, NumSemestr, Zach_Exam,  
    idSubject,idTeacher) VALUES (905,1, 'экзамен', 3,100);
```

Сообщение об ошибке в этом случае будет таким:

### MS SQL Server:

Ошибка: (53,63): Конфликт инструкции INSERT с ограничением FOREIGN KEY "FK\_Sessions\_Teachers\_idTeacher". Конфликт произошел в базе данных "proba", таблица "dbo.Teachers", column 'idTeacher'.

### PostgreSQL:

ОШИБКА: INSERT или UPDATE в таблице "Sessions" нарушает ограничение внешнего ключа "fk\_sess\_tea"

SQL-состояние: 23503

Подробности: Ключ (idTeacher)=(100) отсутствует в таблице "Teachers".

### MySQL:

Cannot add or update a child row: a foreign key constraint fails (`decanat/sessions`, CONSTRAINT `FK\_sessions\_teachers\_idTeacher` FOREIGN KEY (`idTeacher`) REFERENCES `teachers` (`idTeacher`) ON DELETE NO ACTION ON UPDATE NO ACTION)



## 1.4. ВЫБОРКА ДАННЫХ. ОПЕРАТОР SELECT (DQL)

Одной из основных задач, которую решают базы данных – это эффективный поиск необходимых данных. Универсальным подходом для решения этой задачи является применение специального оператора языка SQL - SELECT. Этот оператор достаточно сложный, имеет множество возможностей. Теоретической основой этого оператора является реляционная алгебра, которая доказывает возможность получения с помощью конечного набора операций любых возможных наборов данных.

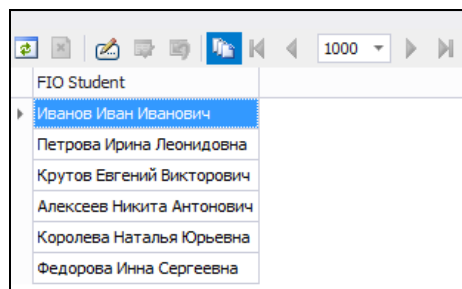
Приведем несколько примеров запросов к таблицам базы данных, демонстрирующих различные возможности языка SQL и различные операции реляционной алгебры. Результаты будем показывать из различных вариантов реализации учебного проекта баз данных (при выполнении на различных СУБД).

Вспомним, что главным отличием синтаксиса команд SQL для PostgreSQL является заключение в кавычки имен таблиц, столбцов и пр. В следующих примерах будем приводить текст запроса в стиле MS SQL Server или MySQL. В случаях более серьезной разницы в записи запросов, будем приводить его текст для каждого СУБД в отдельности.

**Запрос 1. Операция проекции.** Осуществляется выбор только части полей таблицы, т.е. производится вертикальная выборка данных.

Распечатать ФИО всех студентов, зарегистрированных в базе данных:

```
SELECT FIOStudent FROM Students;
```



The screenshot shows a window with a toolbar at the top containing icons for refresh, save, print, and navigation, along with a page size dropdown set to 1000. Below the toolbar is a table with two columns. The first column is titled 'FIO Student' and contains a list of student names. The second column is empty. The first row is highlighted in blue.

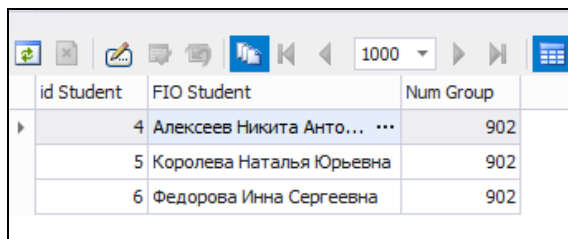
FIO Student	
Иванов Иван Иванович	
Петрова Ирина Леонидовна	
Крутов Евгений Викторович	
Алексеев Никита Антонович	
Королева Наталья Юрьевна	
Федорова Инна Сергеевна	

Рис. 23. Результат выполнения запроса для MS SQL Server.

Запрос 2. **Операция селекции.** Осуществляется горизонтальная выборка – в результат попадают только записи, удовлетворяющие условию.

Распечатать ФИО студентов группы 902.

```
SELECT * FROM Students WHERE NumGroup='902';
```



id Student	FIO Student	Num Group
4	Алексеев Никита Анто...	902
5	Королева Наталья Юрьевна	902
6	Федорова Инна Сергеевна	902

Рис. 24. Результат выполнения запроса для MySQL.

Запрос 3. **Операции соединения.** Здесь следует выделить декартово произведение и на его основе соединение по условию, а также естественное соединение (по одноименным полям или равенству полей с одинаковым смыслом).

Распечатать список зачетов и экзаменов, которые будут сдавать студенты группы 901 в первом семестре.

С помощью декартового произведения и соединения по условию данный запрос запишется так:

```
SELECT TitleSubject, Zach_Exam FROM Sessions, Subjects
WHERE Sessions.NumGroup='901' AND
Sessions.idSubject=Subjects.idSubject AND
Sessions.NumSemestr=1;
```

В этом запросе впервые мы выбираем информацию из нескольких таблиц. В случае использования одноименных полей следует дополнить их именами таблиц согласно схеме ИмяТаблицы.ИмяПоля. В случае же записи для СУБД PostgreSQL каждое из имен должно быть записано в кавычках:

```
SELECT "TitleSubject", "Zach_Exam" FROM "Sessions", "Subjects"
WHERE "Sessions"."NumGroup"='901' AND
"Sessions"."idSubject"="Subjects"."idSubject" AND
"Sessions"."NumSemestr"=1;
```

Панель вывода		
Вывод данных		Построить план выполнения
	TitleSubject text	Zach_Exam text
1	Математический анализ	экзамен
2	Алгебра и геометрия	зачет
3	Алгоритмизация	экзамен
4	Программирование	зачет

Рис. 25. Результат выполнения запроса для PostgreSQL.

Этот же запрос с помощью операции внутреннего соединения имеет следующий вид (соединение производится по равенству одноименных атрибутов idSubject таблиц Sessions и Subjects):

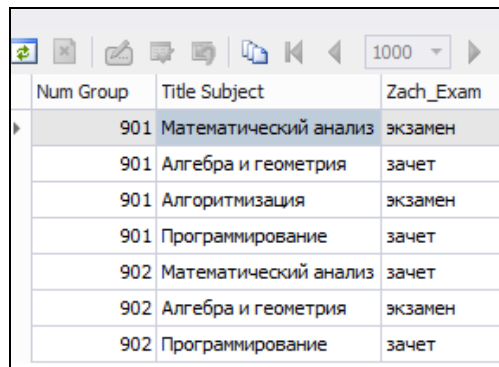
```
SELECT TitleSubject, Zach_Exam FROM Sessions INNER JOIN Subjects
ON Sessions.idSubject=Subjects.idSubject
WHERE Sessions.NumGroup='901' AND Sessions.NumSemestr=1;
```

**Запрос 4. Операция объединения.** Теоретико-множественные операции часто можно записать с помощью логических операций, примененных в конструкции WHERE запроса. Например, нужно получить список зачетов и экзаменов, которые сдают студенты 901 или 902 групп в 1 семестре. Таким образом, нужно объединить два множества, соответствующие двум разным группам. Объединение можно задать с помощью логического ИЛИ.

```
SELECT NumGroup, TitleSubject, Zach_Exam FROM
Sessions INNER JOIN Subjects
ON Sessions.idSubject=Subjects.idSubject
WHERE Sessions.NumSemestr=1 AND
(Sessions.NumGroup='901' OR Sessions.NumGroup='902');
```

Аналогичный результат будет получен с помощью объединения результатов двух запросов (подзапросов) с одинаковой структурой результата:

```
(SELECT NumGroup, TitleSubject, Zach_Exam FROM Sessions
INNER JOIN Subjects ON Sessions.idSubject=Subjects.idSubject
WHERE Sessions.NumSemestr=1 AND Sessions.NumGroup='901')
UNION
(SELECT NumGroup, TitleSubject, Zach_Exam FROM Sessions
INNER JOIN Subjects ON Sessions.idSubject=Subjects.idSubject
WHERE Sessions.NumSemestr=1 AND Sessions.NumGroup='902');
```



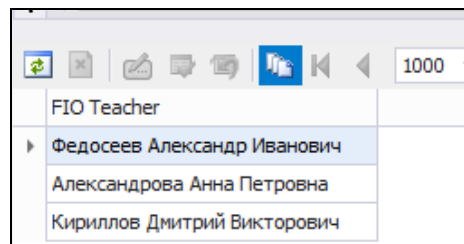
Num Group	Title Subject	Zach_Exam
901	Математический анализ	экзамен
901	Алгебра и геометрия	зачет
901	Алгоритмизация	экзамен
901	Программирование	зачет
902	Математический анализ	зачет
902	Алгебра и геометрия	экзамен
902	Программирование	зачет

Рис. 26. Результат выполнения запроса для MS SQL Server.

Запрос 5. **Операция пересечения.** В простых случаях эту операцию можно описать с помощью логической операции AND. В более сложных случаях эта операция определяется чаще всего с помощью подзапроса и ключевого слова EXISTS, которое показывает наличие похожего элемента во множестве, которое задается подзапросом.

Найти тех преподавателей, которым должны сдавать зачеты или экзамены в первом семестре студенты 901 и 902 групп. Отметим необходимость применения здесь операции переименования (AS) для того, чтобы различить два экземпляра таблицы Sessions (из основного запроса и подзапроса).

```
SELECT FIOTeacher FROM Teachers INNER JOIN Sessions
ON Teachers.idTeacher=Sessions.idTeacher
WHERE Sessions.NumSemestr=1 AND Sessions.NumGroup='901'
AND EXISTS (SELECT * FROM Sessions as s1
WHERE s1.idTeacher=Sessions.idTeacher AND s1.NumSemestr=1
AND s1.NumGroup='902');
```



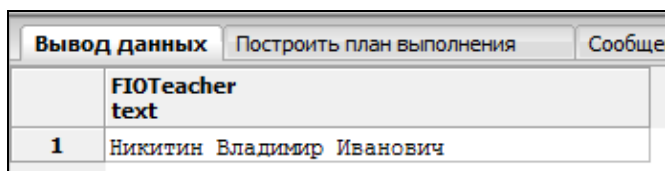
FIO Teacher
Федосеев Александр Иванович
Александрова Анна Петровна
Кириллов Дмитрий Викторович

Рис. 27. Результат выполнения запроса для MySQL.

Запрос 6. **Операция разности.** Эта операция также определяется часто с помощью подзапроса с ключевым словом NOT EXISTS, которое показывает отсутствие элемента во множестве, задаваемом подзапросом. Приведем аналогичный предыдущему пример.

Найти тех преподавателей, которым должны сдавать зачеты или экзамены в первом семестре студенты 901 группы, но не студенты из 902 группы.

```
SELECT FIOTeacher FROM Teachers INNER JOIN Sessions
ON Teachers.idTeacher=Sessions.idTeacher
WHERE Sessions.NumSemestr=1 AND Sessions.NumGroup='901'
AND NOT EXISTS (SELECT * FROM Sessions as s1
WHERE s1.idTeacher=Sessions.idTeacher AND s1.NumSemestr=1
AND s1.NumGroup='902');
```



	FIOTeacher text
1	Никитин Владимир Иванович

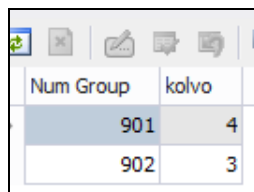
Рис. 28. Результат выполнения запроса для PostgreSQL.

Запрос 7. **Операция группировки.** Эта операция связана со своеобразной сверткой таблицы по полям группировки. Помимо полей группировки результат запроса может содержать итоговые агрегирующие функции по группам (COUNT, SUM, AVG, MAX, MIN).

Найти итоговое количество зачетов и экзаменов, которые должны сдавать студенты различных групп в 1 семестре.

Операция группировки здесь будет применяться к таблице **Sessions**. Поле группировки является номер группы. Агрегирующим полем является количество строк с заданной группой и номером семестра.

```
SELECT NumGroup, COUNT(*) AS kolvo FROM Sessions
WHERE NumSemestr=1 GROUP BY NumGroup;
```



Num Group	kolvo
901	4
902	3

Рис. 29. Результат выполнения запроса для MS SQL Server.

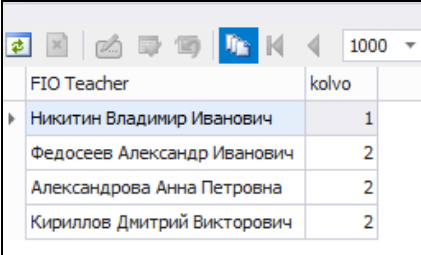
В случае, когда при группировке имеется условие отбора на группу, используется конструкция **HAVING**. Например, в предыдущем примере нам нужно вывести только те группы, в которых количество зачетов-экзаменов равно 3.

```
SELECT NumGroup, COUNT(*) AS kolvo FROM Sessions
WHERE NumSemestr=1 GROUP BY NumGroup HAVING COUNT(*)=3;
```

В СУБД MySQL возможно в условии в конструкции HAVING использовать псевдоним агрегирующего столбца kolvo.

**Запрос 8. Операция сортировки.** Вывести всех преподавателей, которым сдают студенты зачеты-экзамены в первом семестре, в порядке убывания количества зачетов-экзаменов. Для этого следует сначала выбрать нужные элементы таблицы Sessions, затем осуществить естественное соединение полученной таблицы с таблицей Teachers, после чего производится группировка записей в результате запроса и последующая сортировка.

```
SELECT FIOTeacher, COUNT(*) AS kolvo FROM sessions
INNER JOIN teachers
ON sessions.teachers_idTeacher=teachers.idTeacher
WHERE sessions.NumSemestr=1
GROUP BY FIOTeacher ORDER BY kolvo;
```



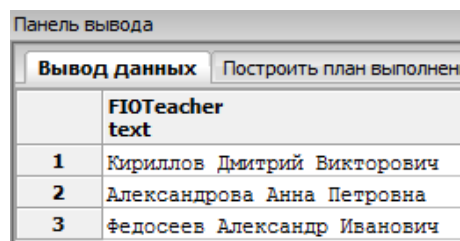
FIO Teacher	kolvo
Никитин Владимир Иванович	1
Федосеев Александр Иванович	2
Александрова Анна Петровна	2
Кириллов Дмитрий Викторович	2

Рис. 30. Результат выполнения запроса для MySQL.

**Запрос 9. Операция деления.** Это самая нетривиальная операция реляционной алгебры, которая обычно применяется тогда, когда требуется найти все записи первой таблицы, которые соединяются естественным образом со всеми записями второй таблицы. Например, нам требуется найти тех преподавателей, которым должны сдать в первом семестре зачеты-экзамены студенты всех групп факультета. Запрос получается достаточно сложный и он связан с выполнением двух операций разности (первая разность - из всевозможных комбинаций групп и преподавателей вычитаются реальные комбинации этих полей, т.е.

результатом становятся всевозможные нереальные пары, вторая разность – выбираются преподаватели, которые в нереальных парах не присутствуют).

```
SELECT FIOTeacher FROM Teachers WHERE idTeacher IN
  (SELECT DISTINCT s0.idTeacher FROM Sessions AS s0
   WHERE NumSemestr=1 AND
   NOT EXISTS (SELECT DISTINCT s1.idTeacher, s2.NumGroup
    FROM Sessions AS s1, Sessions AS s2
    WHERE s1.NumSemestr=1 AND s2.NumSemestr=1
    AND NOT EXISTS (SELECT * FROM Sessions
     AS s3 WHERE s3.idTeacher=s1.idTeacher AND
     s3.NumGroup=s2.NumGroup)
    AND s1.idTeacher=s0.idTeacher));
```



	FIOTeacher text
1	Кириллов Дмитрий Викторович
2	Александрова Анна Петровна
3	Федосеев Александр Иванович

Рис. 31. Результат выполнения запроса для PostgreSQL.

Разберем этот запрос по частям. Все возможные пары «преподаватель» - «группа» получаются с помощью подзапроса:

```
SELECT DISTINCT s1.idTeacher, s2.NumGroup
  FROM Sessions AS s1, Sessions AS s2
 WHERE s1.NumSemestr=1 AND s2.NumSemestr=1
```

добавлением к нему условия:

```
NOT EXISTS (SELECT * FROM Sessions AS s3
  WHERE s3.idTeacher=s1.idTeacher AND s3.NumGroup=s2.NumGroup)
```

из всевозможных пар вычитаются реальные пары, т.е. в результате получаем все возможные нереальные пары «преподаватель»-«группа». Результат этого подзапроса внедряется в другой подзапрос, получающий тех преподавателей, коды которых не присутствуют в этом списке. Далее подключением таблицы Teachers получаем их ФИО.

Наконец, рассмотрим возможность построения **представлений** – виртуальных таблиц, которые представимы как результат выполнения некоторого запроса. Представления в дальнейшем можно использовать в других запросах как таблицы, понимая при этом, что каждый раз при обращении к представлению производится выполнение запроса, по которому представление было создано.

Например, создадим представление, в котором находится информации о том, какие зачеты и экзамены должен сдать в первом семестре каждый студент.

```
CREATE VIEW Student_Session
AS
SELECT FIOStudent, TitleSubject FROM Students INNER JOIN Sessions
      ON Students.NumGroup=Sessions.NumGroup INNER JOIN Subjects
      ON Subjects.idSubject=Sessions.idSubject
WHERE NumSemestr=1;
```

Далее обратимся к этому представлению как к таблице. Например, найти те зачеты-экзамены, которые должен сдать студент Иванов:

```
SELECT TitleSubject FROM Student_Session
      WHERE FIOStudent LIKE 'Иванов%';
```

## 1.5. ХРАНИМЫЕ ПРОЦЕДУРЫ, ФУНКЦИИ И ТРИГГЕРЫ

Хранимые процедуры, функции и триггеры вводятся в базу данных для обеспечения бизнес-логики приложения на уровне серверной его компоненты. Обычно *хранимые процедуры и функции* представляют собой утилиты, которые определенным образом обрабатывают данные или реализуют достаточно сложный алгоритм вычисления некоторых показателей.

*Триггеры* – это частный случай хранимой процедуры, который выполняется автоматически при выполнении команд обновления данных (INSERT, DELETE, UPDATE). Триггеры привязываются к конкретным таблицам базы данных. Для каждой команды должны быть свои триггеры.

В дереве элементов базы данных в любом СУБД имеются группы для определения этих программных элементов:



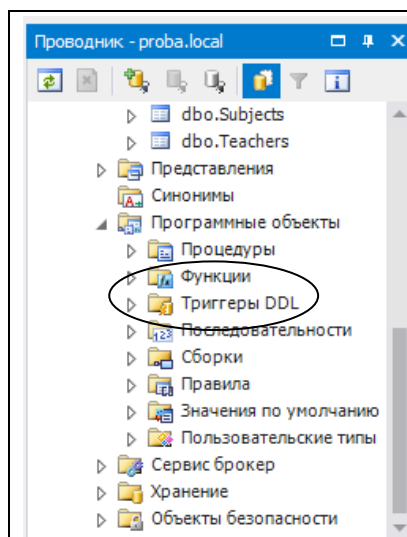


Рис. 32. Дерево элементов в MS SQL Server.

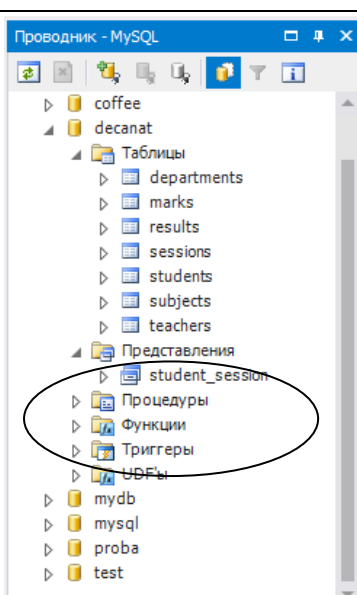


Рис. 33. Дерево элементов в MySQL.

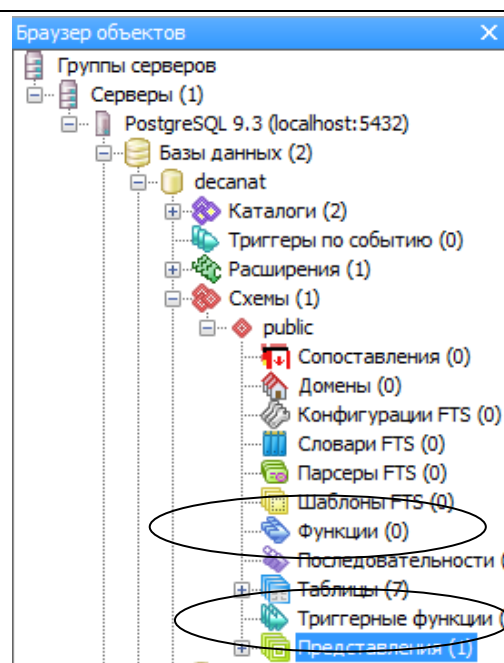


Рис. 34. Дерево элементов в PostgreSQL.

Для создания процедуры, функции или триггера требуется воспользоваться контекстным меню соответствующего элемента дерева. Для создания и редактирования существует специальное диалоговое окно, в котором, в частности, можно задать программный код процедуры, функции или триггера. Программный код формируется посредством перемешивания команд управления и SQL-команд.

Теперь приведем несколько примеров создания хранимых процедур и функций. Здесь мы уже заметим существенные отличия в синтаксисе используемых команд для различных СУБД. Поэтому для каждого из СУБД текст процедур, функций, триггеров и способы вызова укажем отдельно.

**Пример 1.** Напишем хранимую процедуру, которая получает в качестве входного параметра количество баллов и на основании шкалы оценок вычисляет полученную оценку. Результат возвращается через выходной параметр.

**MS SQL Server.** В SQL Server любая переменная именуется, начиная с символа '@'. Остальной код комментировать не требуется.

```
CREATE PROCEDURE dbo.GetMark1 (@ball int, @mark INT OUT)
AS
BEGIN
    IF @ball BETWEEN 55 AND 70
```

```

        SET @mark=3;
    ELSE IF @ball BETWEEN 71 AND 85
        SET @mark=4;
    ELSE IF @ball BETWEEN 86 AND 100
        SET @mark=5;
    ELSE SET @mark=2;
END
GO

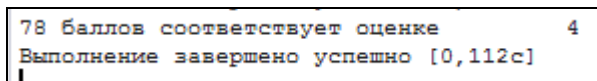
```

Для вызова процедуры требуется создать переменную для применения ее в качестве выходного параметра, после чего воспользоваться командой EXEC. Распечатать результат в выходном потоке можно с помощью оператора PRINT.

```

-- пример вызова процедуры GetMark1
DECLARE @mark INT;
EXEC GetMark1 78, @mark OUT;
PRINT '78 баллов соответствует оценке' + STR(@mark);

```



```

78 баллов соответствует оценке      4
Выполнение завершено успешно [0,112с]

```

Рис. 35. Результат выполнения хранимой процедуры в MS SQL Server.

**MySQL.** Принципиальных отличий в программном коде нет. Стоит отметить, что символ '@' здесь используется только для глобальных переменных, поэтому имена параметров этот символ не имеют.

```

CREATE DEFINER = 'root'@'localhost'
PROCEDURE decanat.GetMark1(in ball INT, out mark INT)
BEGIN
    IF ball BETWEEN 55 AND 70 THEN
        SET mark=3;
    ELSEIF ball BETWEEN 71 AND 85 THEN
        SET mark=4;
    ELSEIF ball BETWEEN 86 AND 100 THEN
        SET mark=5;
    ELSE SET mark=2;
    END IF;
END

```

Вызов процедуры осуществляется следующим образом (достаточно отличным от MS SQL Server). В MySQL не предусмотрено окно сообщений, поэтому вывод осуществляется посредством выборки значения переменной:

```

call GetMark1(89,@m);
select ""+@m as "Оценка";

```

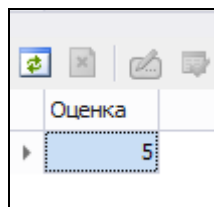


Рис. 36. Результат выполнения хранимой процедуры в MySQL.

**PostgreSQL.** Это СУБД не позволяет создавать процедуры. Здесь используются только функции. Еще одна особенность состоит в том, что функцию можно написать на разных языках. Наиболее распространены sql и plpgsql. Основное отличие языков состоит в том, что в sql доступны только операторы sql, а plpgsql имеет также операторы управления. Интересно, что именовать параметры вовсе не обязательно. К параметрам можно обращаться по номерам, предваренным символом “\$”. Итак, создадим скрипт, в котором запишем следующую функцию:

```
CREATE FUNCTION GetMark1 (integer) RETURNS integer AS $$
DECLARE res INTEGER;
BEGIN
    IF $1 BETWEEN 55 AND 70 THEN
        SELECT 3 INTO res;
    ELSE IF $1 BETWEEN 71 AND 85 THEN
        SELECT 4 INTO res;
    ELSE IF $1 BETWEEN 86 AND 100 THEN
        SELECT 5 INTO res;
    ELSE SELECT 2 INTO res;
    END IF; END IF; END IF;
    RETURN res;
END;
$$ LANGUAGE plpgsql;
```

Отметим применение символов “\$\$” в начале и конце функции. Они позволяют игнорировать символы-разделители внутри этих своеобразных скобок. Функция декларирует тип возвращаемого значения с помощью ключевого слова RETURNS. В теле функции создается переменная для хранения результата, которой присваивается значение в зависимости от ветки условных операторов, по которой пойдет управление. К параметру производится обращение посредством “\$1”. Отметим еще, что в конце следует указать используемый язык написании функции.

Вызов функции:

```
select GetMark1(68);
```

**Пример 2.** Чтобы при смене правил вычисления оценок не нужно было бы менять процедуру, мы создали справочную таблицу для хранения всех оценок и их диапазонов Marks. Пришло время ею воспользоваться. Второй вариант функции получения оценки по набранным баллам будет обращаться к этой таблице за информацией.

Оформим этот вариант в виде функции с одним параметром, хранящим набранные баллы, и возвращающую найденную оценку или 2 в случае, когда набранным баллам ничего в таблице не соответствует. В данной функции демонстрируется использование переменных, запросов и условного оператора. Приведем два варианта функции. С помощью запроса на количество таких записей алгоритм первого варианта предусматривает определение, есть ли в таблице соответствующая баллам оценка. Второй вариант пользуется специальной функцией EXISTS, которая, принимая в качестве аргумента запрос, возвращает логическое значение, определяющее, есть ли в результате запроса записи.

**MS SQL Server.** В предыдущем примере мы уже видели одно из отличий языка для MS SQL Server, связанное с присвоением значений переменным. В MS SQL Server для этого предназначен оператор SET. В других рассматриваемых нами СУБД в этой роли выступает оператор SELECT.

Итак, первый вариант функции. В нем определяются две переменные для хранения количества записей и оценки. Значением оценки по умолчанию является оценка 2. После определяется количество записей в таблице Marks, соответствующее набранным баллам и, если это количество больше 0, найденная оценка присваивается переменной @mark, которая в конце возвращается как результат функции. Отметим, что оператор RETURN должен быть последним в функции.

```
CREATE FUNCTION dbo.GetMark2 (@ball int)
RETURNS INT
AS
BEGIN
    DECLARE @kolvo INT, @mark INT;
    SET @mark=2;
    SET @kolvo=(SELECT COUNT(*) FROM Marks WHERE
```

```

                                @ball between LowBalls and HighBalls);
IF @kolvo>0
    SET @mark=(SELECT idMark FROM Marks WHERE
                                @ball between LowBalls and HighBalls);
RETURN @mark;
END
GO

```

Во втором варианте функции в условном операторе вместо переменной @kolvo используется вызов функции EXISTS.

```

CREATE FUNCTION dbo.GetMark3(@ball int)
RETURNS INT
AS
BEGIN
    DECLARE @mark INT;
    SET @mark=2;
    IF EXISTS(SELECT * FROM Marks WHERE @ball
                                between LowBalls and HighBalls)
        SET @mark=(SELECT idMark FROM Marks WHERE
                                @ball between LowBalls and HighBalls);
    RETURN @mark;
END
GO

```

Вызов функции оформляется следующим образом:

```

DECLARE @mark INT;
SET @mark=dbo.GetMark3(93);
PRINT '93 балла соответствует оценке' + STR(@mark);

```

**MySQL.** Аналогичные функции для сервера MySQL определяются следующим образом:

```

CREATE FUNCTION decanat.GetMark2(ball int)
RETURNS int(11)
BEGIN
    DECLARE kolvo, mark INT;
    SELECT 2 INTO mark;
    SELECT COUNT(*) INTO kolvo FROM Marks WHERE
                                ball between LowBalls and HighBalls;
    IF kolvo>0 THEN
        SELECT idMark INTO mark FROM Marks WHERE
                                ball between LowBalls and HighBalls;
    END IF;
    RETURN mark;
END

```

```

CREATE FUNCTION decanat.GetMark3(ball int)
RETURNS int(11)
BEGIN
    DECLARE mark INT;
    SELECT 2 INTO mark;
    IF EXISTS (SELECT * FROM Marks WHERE
                                ball between LowBalls and HighBalls) THEN

```

```

        SELECT idMark INTO mark FROM Marks WHERE
                                ball between LowBalls and HighBalls;
    END IF;
    RETURN mark;
END

```

Вызов функции можно осуществлять непосредственно в выражении, например:

```
SELECT ""+GetMark2(89) as "Оценка";
```

**PostgreSQL.** Как уже было сказано, в PostgreSQL хранимых процедур нет, в этом СУБД используются только функции. Программный код практически не будет отличаться от кода для MySQL за исключением определения переменных за скобками функции, обращения к параметру и обращения к полям и таблице базы данных:

```

CREATE FUNCTION GetMark2 (integer) RETURNS integer AS $$
DECLARE kolvo INTEGER;
DECLARE mark INTEGER;
BEGIN
    SELECT 2 INTO mark;
    SELECT COUNT(*) INTO kolvo FROM "Marks"
                                WHERE $1 between "LowBalls" and "HighBalls";
    IF kolvo>0 THEN
        SELECT "idMark" INTO mark FROM "Marks"
                                WHERE $1 between "LowBalls" and "HighBalls";
    END IF;
    RETURN mark;
END;
$$ LANGUAGE plpgsql;

```

```

CREATE FUNCTION GetMark3 (integer) RETURNS integer AS $$
DECLARE mark INTEGER;
BEGIN
    SELECT 2 INTO mark;
    IF EXISTS (SELECT * FROM "Marks"
                                WHERE $1 between "LowBalls" and "HighBalls") THEN
        SELECT "idMark" INTO mark FROM "Marks"
                                WHERE $1 between "LowBalls" and "High-
Balls";
    END IF;
    RETURN mark;
END;
$$ LANGUAGE plpgsql;

```

Теперь приведем примеры создания триггеров.

**Пример 1.** Создадим триггер для вставки в таблицу результатов сессии, в котором проверяются ограничения целостности (студент с заданным кодом существует, предмет с заданным кодом существует, дисциплину нужно сдавать

именно в этом семестре). Если произойдет нарушение этих ограничений, то требуется откатить транзакцию, т.е. не выполнять вставку записи. Если же все данные будут корректными, проведем заполнение значений полей даты сдачи зачета/экзамена как текущей и вычислим оценку по указанным баллам.

Для проверки корректности данных для вставки создадим вспомогательную хранимую функцию, чтобы код триггера был не очень сложным. (Для некоторых версий СУБД требуется, чтобы в триггере было упоминание только текущей записи, обращение к другим таблицам и записям недоступно).

### **MS SQL Server:**

```
CREATE FUNCTION dbo.IsCorrect(@idStud INT, @idSubj INT,
                              @Sem INT, @idTeach INT) RETURNS INT
AS
BEGIN
    IF EXISTS (SELECT * FROM Students INNER JOIN Sessions
               ON Students.NumGroup=Sessions.NumGroup
               INNER JOIN Subjects ON
               Sessions.idSubject=Subjects.idSubject
               INNER JOIN Teachers ON
               Sessions.idTeacher=Teachers.idTeacher
               WHERE Students.idStudent=@idStud AND
               Subjects.idSubject=@idSubj
               AND Teachers.idTeacher=@idTeach and NumSemestr=@Sem)
        RETURN 1;
    RETURN 0;
END
GO
```

### **MySQL:**

```
CREATE FUNCTION IsCorrect (idStud INT, idSubj INT, Sem INT, idTeach INT)
    RETURNS INT(11)
BEGIN
    RETURN EXISTS (SELECT * FROM Students INNER JOIN Sessions
                   ON Students.NumGroup=Sessions.NumGroup
                   INNER JOIN Subjects ON
                   Sessions.idSubject=Subjects.idSubject
                   INNER JOIN Teachers ON
                   Sessions.idTeacher=Teachers.idTeacher
                   WHERE Students.idStudent=idStud AND
                   Subjects.idSubject=idSubj
                   AND Teachers.idTeacher=idTeach and NumSemestr=Sem);
END
```

### **PostgreSQL:**

```
CREATE FUNCTION IsCorrect(integer, integer, integer, integer)
    RETURNS BOOLEAN AS $$
BEGIN
    RETURN EXISTS (SELECT * from "Students" INNER JOIN "Sessions"
                   ON "Students"."NumGroup"="Sessions"."NumGroup"
                   INNER JOIN "Subjects" ON
                   "Sessions"."idSubject"="Subjects"."idSubject"
```

```

INNER JOIN "Teachers" ON
"Sessions"."idTeacher"="Teachers"."idTeacher"
WHERE "Students"."idStudent"=$1 AND
"Subjects"."idSubject"=$2
AND "Teachers"."idTeacher"=$3 AND "NumSemestr"=$4);
END;
$$ LANGUAGE plpgsql;

```

Триггер на вставку записи в таблицу Results будет вызывать функцию проверки корректности, передавая в функцию поля из новой записи. Если запись будет корректной, будут скорректированы поля оценки и даты сдачи зачета/экзамена. В противном случае должен быть произведен откат транзакции.

### MS SQL Server:

При вставке записи сначала запись попадает в виртуальную таблицу inserted (при удалении будет использоваться таблица deleted, при изменении записи используются обе таблицы – в inserted хранятся новые значения записи, в deleted – прежние значения полей записи). Поэтому сначала получаем данные новой записи из таблицы inserted, после чего проверяем их на корректность. В случае корректных данных оставшиеся поля (дата и оценка) изменяются посредством команды UPDATE. Откат транзакции в случае некорректных данных производится с помощью команды ROLLBACK.

```

CREATE TRIGGER trigger1
ON dbo.Results
FOR INSERT
AS
BEGIN
-- объявление необходимых переменных для хранения данных новой записи
DECLARE @idStudent INT, @idSubject INT,
        @idTeacher INT, @NumSemestr INT, @Balls INT;

-- чтение данных новой записи
SET @idStudent =(SELECT idStudent FROM inserted);
SET @idSubject =(SELECT idSubject FROM inserted);
SET @idTeacher =(SELECT idTeacher FROM inserted);
SET @NumSemestr =(SELECT NumSemestr FROM inserted);
SET @Balls =(SELECT Balls FROM inserted);

-- проверка на корректность данных
IF dbo.IsCorrect(@idStudent, @idSubject, @NumSemestr, @idTeacher)=0
BEGIN
-- данные некорректны. Выводим сообщение об ошибке
-- и производим откат транзакции
PRINT 'Ошибка данных: данные некорректны';
ROLLBACK;
END
ELSE
-- изменение полей даты и вычисление оценки.

```



```

-- В условии указывается первичный ключ
UPDATE dbo.Results SET mark=dbo.GetMark3(@Balls), DateExam=GETDATE()
    WHERE idStudent=@idStudent AND idSubject=@idSubject
    AND idTeacher=@idTeacher AND NumSemestr=@NumSemestr;

END
GO

```

## MySQL:

Для MySQL данный триггер запишется проще, так как здесь проще получить данные новой записи. Новая запись хранится в виде объекта New (запись при удалении хранится в виде объекта Old). Однако имеется проблема, связанная с отсутствием команды отката триггера. В этом случае рекомендуется выполнить какую-нибудь ошибочную команду, например, вставить запись с уже существующим ключом. Ошибка в этой команде приведет к отмене действий всей транзакции (команды и триггера).

```

CREATE TRIGGER decanat.trigger1
    BEFORE INSERT
    ON decanat.results
    FOR EACH ROW
BEGIN
    IF IsCorrect(New.idStudent, New.idSubject, New.NumSemestr, New.idTeacher)
    THEN
        SET New.Mark=GetMark3(New.Balls);
        SET New.DateExam=Now();
    ELSE
        insert into Departments values (1,"","");
    END IF;
END

```

## PostgreSQL:

В PostgreSQL триггер как таковой связан со специальной триггерной функцией, в которой и осуществляется вся обработка данных. Триггерная функция возвращает объект-запись (NEW или OLD), с которой производится работа. При написании триггера мы указываем только для какой операции, для какой таблицы и каков тип триггера, после чего вызываем триггерную функцию. Откат производится генерацией исключительной ситуации с указанием сообщения об ошибке. В остальном код похож на тот, что писался для MySQL:

```

-- Создание триггерной функции на вставку результата сдачи экзамена
CREATE FUNCTION trigger_results_insert() RETURNS trigger AS $$
BEGIN
    IF IsCorrect(NEW."idStudent", NEW."idSubject",
        NEW."idTeacher", NEW."NumSemestr")
    THEN
        SELECT GetMark3(NEW."Balls") INTO NEW."Mark";
    ELSE
        RAISE EXCEPTION 'Invalid data for trigger_results_insert';
    END IF;
END;

```

```

        SELECT Now() INTO New."DateExam";
    ELSE
        -- генерация исключительной ситуации
        RAISE EXCEPTION 'Ошибка корректности данных';
    END IF;
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

-- Создание триггера на вставку нового результата экзамена
CREATE TRIGGER tr_results_insert
BEFORE INSERT ON "Results" FOR EACH ROW
EXECUTE PROCEDURE trigger_results_insert();

```

Для проверки работы триггера (например, для MySQL) проведем следующие операции вставки:

```

INSERT INTO Results (idStudent, idSubject, idTeacher, NumSemestr, Balls)
VALUES (1,1,1,1,78);
INSERT INTO Results (idStudent, idSubject, idTeacher, NumSemestr, Balls)
VALUES (2,1,1,1,98);
INSERT INTO Results (idStudent, idSubject, idTeacher, NumSemestr, Balls)
VALUES (6,1,1,1,68);

```

Согласно данным, которые мы вносили в таблицу, последняя запись не должна быть добавлена.

**Пример 2.** Приведем еще один пример триггера на вставку новой записи в таблицу результатов. Этот триггер должен срабатывать после вставки и быть связан с подсчетом рейтинга студентов. Триггеры «после» часто используются для проведения специальной обработки данных на основании выполненной операции и могут быть связаны с другими таблицами.

Для этого введем в базу данных новую таблицу, например, с помощью следующей SQL-команды:

```

CREATE TABLE Reyting
( idStudent INT PRIMARY KEY,
  summ_balls INT,
  CONSTRAINT fk_reyting
    FOREIGN KEY (idStudent) REFERENCES Students (idStudent)
)

```

При вставке нового результата рейтинг студента должен меняться. Таким образом, нужно проанализировать, есть ли запись о студенте – в случае положительного ответа произвести суммирование баллов, иначе добавить новую запись в таблицу рейтинга.

## MS SQL Server:

```
CREATE TRIGGER trigger2
ON dbo.Results
AFTER INSERT
AS
BEGIN
    DECLARE @idStudent INT, @Balls INT;
    SET @idStudent = (SELECT idStudent FROM inserted);
    SET @Balls =(SELECT Balls FROM inserted);

    IF EXISTS(SELECT * FROM Reyting WHERE idStudent=@idStudent)
        UPDATE Reyting SET summ_balls=summ_balls+@Balls
                        WHERE idStudent=@idStudent;
    ELSE
        INSERT INTO Reyting (idStudent, summ_balls)
                        VALUES (@idStudent, @Balls);
END
GO
```

## MySQL:

```
CREATE TRIGGER decanat.trigger2
AFTER INSERT
ON decanat.results
FOR EACH ROW
BEGIN
    IF EXISTS(SELECT * FROM Reyting WHERE idStudent=new.idStudent) THEN
        UPDATE Reyting SET summ_balls=summ_balls+new.Balls
                        WHERE idStudent=new.idStudent;
    ELSE
        INSERT INTO Reyting (idStudent, summ_balls)
                        VALUES (New.idStudent, New.balls);
    END IF;
END
```

## PostgreSQL:

```
CREATE FUNCTION trigger_results_insert_after() RETURNS trigger AS $$
BEGIN
    IF EXISTS(SELECT * FROM "Reyting" WHERE "idStudent"=NEW."idStudent") THEN
        UPDATE "Reyting" SET "summ_balls"="summ_balls"+NEW."Balls"
                        WHERE "idStudent"=NEW."idStudent";
    ELSE
        INSERT INTO "Reyting" ("idStudent", "summ_balls")
                        VALUES (NEW."idStudent", NEW."Balls");
    END IF;
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER tr_results_insert_after
AFTER INSERT ON "Results" FOR EACH ROW
EXECUTE PROCEDURE trigger_results_insert_after();
```

Поэкспериментируйте сами со вставками записей, чтобы изменялся рейтинг студентов.

Разберем еще один пример хранимой функции для демонстрации использования **курсоров** – временных таблиц, представляющих собой результаты выполнения запроса и обрабатываемые построчно – от первой записи до последней. Для этого создадим еще одну версию функции перевода баллов в оценку – каждая строка таблицы Marks в ней будет обрабатываться построчно до получения строки с нужной оценкой или отсутствием соответствующей оценки.

### MS SQL Server:

```
CREATE FUNCTION dbo.GetMark4(@balls INT)
RETURNS INT
BEGIN
    DECLARE @res INT, @mark INT, @lowB INT, @highB INT;
    SET @res=2;

    -- декларация курсора, связанного с запросом
    DECLARE mark_cursor CURSOR FOR SELECT * FROM Marks;

    -- открытие курсора
    OPEN mark_cursor;
    -- считывание первой строки курсора в переменные @mark, @lowB, @highB
    FETCH NEXT FROM mark_cursor INTO @mark, @lowB, @highB;

    -- цикл продолжается, пока считывание возможно,
    -- на это укажет глобальная переменная
    WHILE @@FETCH_STATUS = 0
    BEGIN
        -- определяем, соответствуют ли баллы текущей оценке
        IF @balls BETWEEN @lowB AND @highB
        BEGIN
            SET @res=@mark;
            BREAK;
        END
        -- переход к следующей строке курсора
        FETCH NEXT FROM mark_cursor INTO @mark, @lowB, @highB;
    END

    -- закрытие курсора
    CLOSE mark_cursor;
    -- разрушение курсора
    DEALLOCATE mark_cursor;
    RETURN @res;
END
GO
```

**MySQL:** для обработки завершения записей курсора здесь требуется создать специальный обработчик CONTINUE HANDLER FOR NOT FOUND. В остальном работа с курсором аналогична.

```
CREATE DEFINER = 'root'@'localhost'
FUNCTION decanat.GetMark4(balls INT)
RETURNS int(11)
BEGIN
    -- переменные для хранения полей кортежа из таблицы Marks
    DECLARE mark, lowB, highB, res INT;
```

```

-- переменная для определения, завершен ли просмотр курсора
DECLARE is_end INT DEFAULT 0;
-- определение курсора для таблицы Marks
DECLARE mark_cursor CURSOR FOR SELECT * FROM Marks;

-- объявление обработчика ошибки завершения записей курсора
DECLARE CONTINUE HANDLER FOR NOT FOUND SET is_end=1;

SET res=2;
-- открытие курсора
OPEN mark_cursor;
-- считывание первой записи курсора
FETCH mark_cursor INTO mark, lowB, highB;

-- организация цикла просмотра строк из курсора
WHILE is_end=0 DO

    -- проверка диапазона баллов для текущей оценки
    IF balls BETWEEN lowB AND highB THEN
        SET res=mark;
        -- организация выхода из цикла
        SET is_end=1;
    END IF;
    -- считывание очередной записи курсора
    FETCH mark_cursor INTO mark, lowB, highB;
END WHILE;
CLOSE mark_cursor;
RETURN res;
END

```

**PostgreSQL:** как и в предыдущем случае отличия будут в организации цикла и выхода из него.

```

CREATE FUNCTION GetMark4 (integer) RETURNS integer AS $$
DECLARE res integer;
DECLARE mark integer;
DECLARE lowB integer;
DECLARE highB integer;
DECLARE mark_cursor CURSOR FOR SELECT * FROM "Marks";
BEGIN
    res:=2;
    OPEN mark_cursor;--открываем курсор
    LOOP --начинаем цикл по курсору
        --извлекаем данные из строки и записываем их в переменные
        FETCH mark_cursor INTO mark, lowB, highB;
        --если такого периода и не возникнет, то мы выходим
        IF NOT FOUND THEN EXIT;END IF;
        IF $1 BETWEEN lowB AND highB THEN
            res:=mark;
        END IF;
    END LOOP;--заканчиваем цикл по курсору
    CLOSE mark_cursor; --закрываем курсор
    return res;
END;
$$ LANGUAGE plpgsql;

```

## **Часть II. КЛИЕНТСКИЕ ТЕХНОЛОГИИ**

Очевидно, что конечный пользователь не знает языка SQL, поэтому ему нужно предоставить удобный и понятный пользовательский интерфейс, с помощью которого он мог бы формировать и получать данные. В архитектуре «клиент-сервер» СУБД представляется серверной стороной, а в качестве клиентского уровня может использоваться любое приложение (консольное, оконное, web-приложение), разработанное с помощью различных технологий и языков программирования. Общая концепция обмена данными между клиентом и сервером по большей степени не зависит от технологии реализации клиентского приложения. Поэтому разберем вопросы, связанные с разработкой клиентского приложения на языке программирования C# с использованием технологии ADO.NET. Приведем типовые примеры решения различных задач, возникающих на клиентской части приложения.

Для подключения к базам данных в среде .NET Framework используется пространство имен System.Data и его подпространства имен (OleDb, Odbc и пр.). Разберем основные особенности работы элементов этого пространства имен. В основе любого из них – драйвер баз данных (коннектор) – специальное программное обеспечение, которое является посредником между клиентским приложением и базой данных (серверной или файловой). Для обращения используются стандартные технологии Ole Db или ODBC.

### **2.1. ВЫПОЛНЕНИЕ ЗАПРОСА К БАЗЕ ДАННЫХ ИЗ КЛИЕНТСКОГО ПРИЛОЖЕНИЯ**

Простая модель обращения к базе данных реализуется через три основных класса, которые могут быть реализованы на основе как универсальных технологий (Ole Db или ODBC), так и конкретных СУБД (например, SQL Server или MySQL):

- Класс соединения;
- Класс команды;
- Класс курсора для получения данных из результата запроса.

Класс соединения настраивает параметры базы данных, к которой следует подключиться. Открытие такого соединения позволяет формулировать команды к серверу баз данных. Команды оформляются с помощью языка SQL и оформляются в программе с помощью класса команды. Класс курсора предназначен для чтения данных результата запроса. Курсор обычно является последовательным, т.е. с помощью него мы можем получить последовательно все записи результата запроса, начиная с первой до последней, но не можем вернуться к уже просмотренным записям.

Настройка соединения осуществляется через понятие строки соединения – параметров, которые описывают драйверы СУБД, конкретные базы данных и параметры безопасности для подключения к базе.

Например, для настройки источника данных ODBC требуется воспользоваться средством администрирования операционной системы Windows.

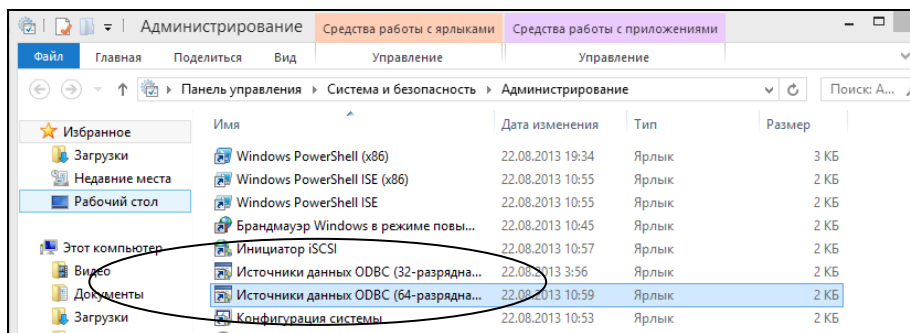


Рис. 37. Панель администрирования ОС Windows.

Для общения с серверными СУБД требуется, чтобы источник данных был системным:

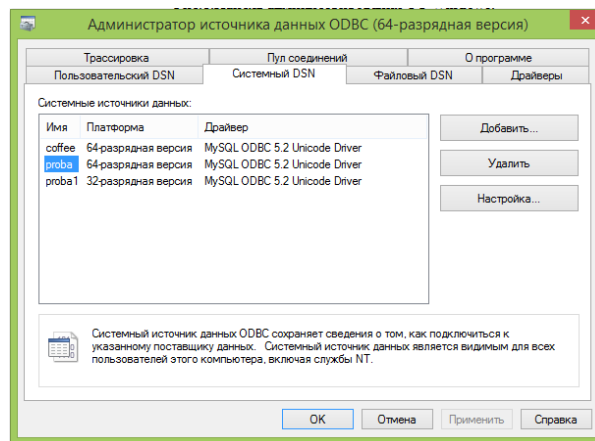


Рис. 38.Окно просмотра источников данных ODBC.

При создании и редактировании источника данных требуется выбрать драйвер и задать параметры подключения к базе данных:

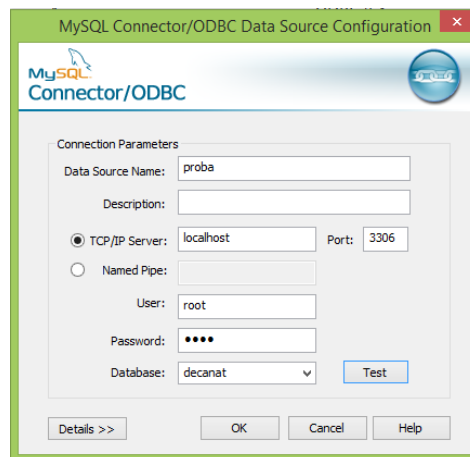


Рис. 39.Окно настройки источника данных для сервера MySQL.

Драйверы могут устанавливаться и регистрироваться в операционной системе вместе с соответствующей СУБД, а могут быть и самостоятельными компонентами, которые следует установить отдельно. Так, например, для SQL Server драйвер SQL Native Client устанавливается в процессе установки СУБД, а для сервера MySQL драйвер (MySQL ODBC Connector) является дополнительной компонентой, которую следует установить отдельно.

Для настройки строки соединения через технологию Ole Db можно воспользоваться специальной утилитой операционной системы Windows. Для этого требуется создать файл с расширением “.udl”. При последующем открытии этого файла будут вызвано окно настройки источника данных:



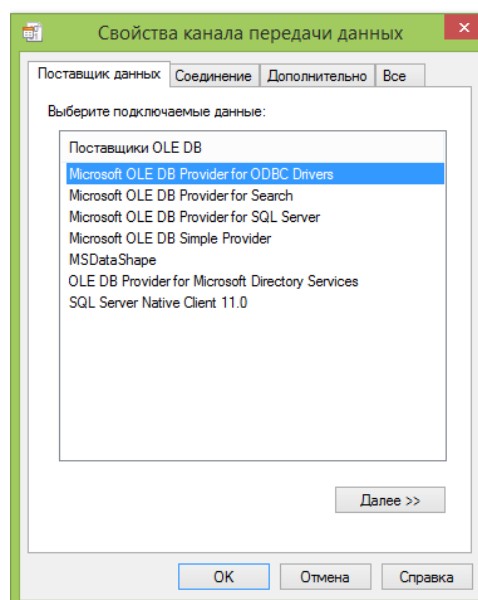


Рис. 40. Окно сервиса OLE DB Core Services.

После выбора поставщика данных на вкладке «Подключение» следует настроить параметры базы данных (имя, местоположение, параметры безопасности). Этот файл можно будет открыть с помощью текстового редактора – в нем будет записана строка соединения через выбранный провайдер Ole Db.

Для примера рассмотрим подключение к базе данных на SQL Server. Создадим файл connect.udl (например, с помощью «Блокнота») и откроем его с помощью службы OLE DB Core Services. На вкладке «Поставщик данных» следует выбрать пункт «SQL Server Native Client номер версии» (номер версии, очевидно, зависит от установленного на компьютере SQL Server'a). Далее на вкладке «Соединение» следует настроить параметры подключения к серверу – имя сервера, параметры входа (обычно устанавливаются из системы безопасности операционной системы), имя используемой базы данных. С помощью кнопки «Проверить подключение» можно протестировать созданную строку соединения.

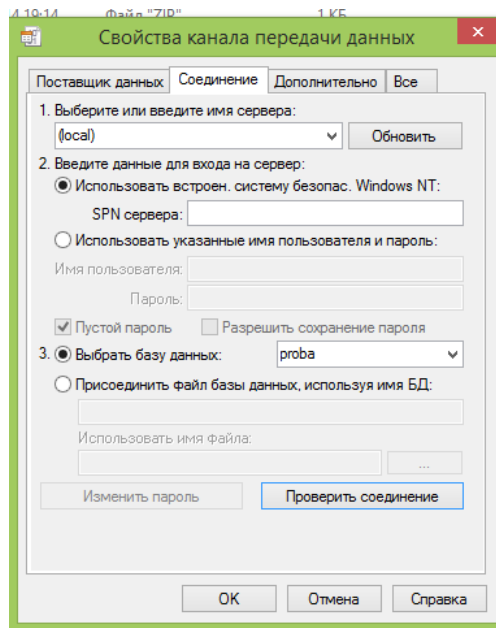


Рис. 41. Окно настройки параметров подключения к SQL Server.

Далее при открытии файла connect.udl с помощью «Блокнота» мы увидим сгенерированную строку подключения:

```
Provider=SQLNCLI11.1;Integrated Security=SSPI;
Persist Security Info=False;
User ID="";Initial Catalog=proba;Data Source=(local);
Initial File Name="";Server SPN=""
```

Далее приведен программный код, в котором осуществляется подключение к источнику данных, заданному с помощью ODBC и получение данных из базы данных с помощью формирования SQL-запроса:

```
// создание подключения к базе данных на основе строки соединения
// с указанием источника данных ODBC
OdbcConnection con = new OdbcConnection("DSN=proba");
// подключение к источнику данных
con.Open();
// формирование команды SQL на выборку данных
OdbcCommand com = new OdbcCommand("select * from Session", con);
// выполнение команды на сервере и сохранение результата
// в курсоре типа OdbcDataReader
OdbcDataReader dr=com.ExecuteReader();
// переход к следующей строке посредством функции Read()
// пока строки в результате есть - печатаем информацию из строк
while (dr.Read())
    Console.WriteLine("\""+dr["NumGroup"]+" "+dr["idSubject"]+
        "\""+dr["Zach_exam"]);
// закрывается соединение
dr.Close();
con.Close();
```

Аналогичным будет программный код и в случае использования подключения по технологии Ole DB. Отличия будут только в именах используемых классов для доступа к базе данных (OleDbConnection, OleDbCommand, OleDbDataReader) и, соответственно, в пространстве имен, содержащее эти классы.

Отметим еще момент, связанный с обеспечением доступа к полям строки курсора. Текущая строка представляется как ассоциативный массив, доступ к элементам которого можно осуществлять по номерам или по именам столбцов, что очень удобно с точки зрения последующего чтения программного кода.

## 2.2. ПАРАМЕТРЫ ЗАПРОСА

Нередко запрос зависит от параметров, являющиеся значениями, по которым осуществляется выборка или другие фрагменты запроса, например, условия, формируемые программным образом. В этом случае используется коллекция параметров, которая имеется у объекта класса OdbcCommand. При создании параметра в запросе указывается символ "?", а далее добавляется применяемое значение параметра в коллекцию параметров SQL-команды. В том случае, когда базовой технологией доступа является Ole Db, параметры в запросе именуются, причем эти имена начинаются с символа "@".

Пусть, например, номер группы, для которой требуется распечатать план сессии, вводится с клавиатуры и становится условием выборки записей из таблицы плана сессии для студентов.

```
// создание подключения к базе данных на основе строки соединения
// с указанием источника данных ODBC
OdbcConnection con = new OdbcConnection("DSN=proba");
// подключение к источнику данных
con.Open();
// ввод номера группы
string group = Console.ReadLine();
// формирование команды SQL на выборку данных
OdbcCommand com = new OdbcCommand
    ("select * from Sessions where NumGroup=?",
con);
```

```

// задание значение параметра запроса
com.Parameters.AddWithValue("@par",group) ;

// выполнение команды на сервере и сохранение результата
// в курсоре типа OdbcDataReader
OdbcDataReader dr=com.ExecuteReader();
// переход к следующей строке посредством функции Read()
// пока строки в результате есть - печатаем информацию из строк
while (dr.Read())
    Console.WriteLine(" "+dr["NumGroup"]+" "+dr["idSubject"]+
                      " "+dr["Zach_exam"]);

// закрывается соединение
dr.Close();
con.Close();

```

Нередко использование параметров применяется при удаленном вызове хранимых процедур и функций. Вспомним, что у нас имеются функции, которые позволяют перевести баллы в оценку (например, GetMark1). Вызов этой функции может быть таким:

```

// создание подключения к базе данных на основе строки соединения
// с указанием источника данных ODBC
OdbcConnection con = new OdbcConnection("DSN=proba");
// подключение к источнику данных
con.Open();
// ввод набранных баллов на экзамене
string b = Console.ReadLine();
// формирование команды SQL на вызов функции и получение ее результата
OdbcCommand com = new OdbcCommand("select GetMark1(?)", con);

// задание значение параметра запроса
com.Parameters.AddWithValue("@ball",b) ;

// выполнение команды на сервере и сохранение результата
// в курсоре типа OdbcDataReader
OdbcDataReader dr=com.ExecuteReader();
// переход к первой строке - результат вызова функции
dr.Read();
Console.WriteLine("Оценка - "+dr[0]);
// закрывается соединение
dr.Close();
con.Close();

```

## 2.3. ВЫПОЛНЕНИЕ КОМАНД DML

Напомним, что командами DML являются команды вставки новых записей, изменения существующих записей и удаления записей. Данные команды возвращают число – количество строк, с которыми была выполнена требуемая

операция. Вызов этих команд из клиентского приложения отличается только функцией класса OdbcCommand (или OleDbCommand) – вместо ExecuteReader() вызывается функция ExecuteScalar().

Например, пусть создается новая учебная дисциплина:

```
// создание подключения к базе данных на основе строки соединения
// с указанием источника данных ODBC
OdbcConnection con = new OdbcConnection("DSN=proba");
// подключение к источнику данных
con.Open();
// ввод названия новой учебной дисциплины
string title = Console.ReadLine();
// формирование команды SQL на добавление данных – в таблице ключ
// задается с помощью поля-счетчика, так что указывать
// его в запросе на вставку не обязательно
OdbcCommand com = new OdbcCommand
    ("insert into Subjects values ('?')", con);
// задание значение параметра запроса
com.Parameters.AddWithValue("@par", title);
// выполнение команды на сервере
com.ExecuteNonQuery();
// закрывается соединение
con.Close();
```

## 2.4. ПОНЯТИЕ НАБОРА ДАННЫХ КАК ВИРТУАЛЬНОЙ БАЗЫ ДАННЫХ

В пространстве имен System.Data определена система классов, которая реализует понятие набора данных. Набор данных (DataSet) представляет собой виртуальную базу данных, которая расположена в оперативной памяти и состоит из набора (коллекции) таблиц (объектов класса DataTable) и связей между ними (объектов класса DataRelation). Каждая таблица имеет набор столбцов (объектов типа DataColumn) и набор строк (объектов типа DataRow). Все таблицы в наборе данных хранятся в виде коллекции Tables. Столбцы в таблице хранятся тоже в виде коллекции под названием Columns, а строки – в виде коллекции под названием Rows. Обращение с таблицами набора данных напоминает работу с базой данных – у таблиц имеется функция Select(), которая задает параметры сортировки и выборки данных. Таким образом, можно моделировать различные запросы.

Набор данных можно создавать программно посредством создания объектов-таблиц и столбцов. Сохранять информацию набора данных можно в формате XML с помощью функции WriteXml(). Также возможна и загрузка информации из XML-файла с помощью функции ReadXml().

## 2.5. СВЯЗЬ НАБОРА ДАННЫХ И БАЗЫ ДАННЫХ

Набор данных может быть связан с реальными базами данных, реализованными с помощью различных СУБД. Это реализуется с помощью объекта типа OdbcDataAdapter (или OleDbDataAdapter в зависимости от способа соединения с базой данных). Создание SQL-команды осуществляется с помощью адаптера. Кроме того, класс-адаптер имеет специальный метод, который позволяет записать результат выполнения запроса в таблицу набора данных.

Приведем пример заполнения набора данных с помощью адаптера:

```
// создание подключения к базе данных на основе строки соединения
// с указанием источника данных ODBC
OdbcConnection con = new OdbcConnection("DSN=proba");
// подключение к источнику данных
con.Open();
// формирование адаптера для связи с базой данных
OdbcDataAdapter adapt = new OdbcDataAdapter
    ("select * from Sessions", con);
// создание набора данных для сохранения результата запроса
DataSet ds = new DataSet();
// заполнение таблицы набора данных (второй параметр)
// из базы данных через адаптер
adapt.Fill(ds, "Sessions");
// перебор строк из полученной таблицы Sessions набора данных
// имена столбцов в таблице набора данных будут теми же,
// что и в исходной таблице базы данных
foreach (DataRow dr in ds.Tables["Sessions"])
    Console.WriteLine("\""+dr["NumGroup"]+" "+dr["idSubject"]+
        "\""+dr["Zach_exam"]);
con.Close();
```

Отметим, что, так как набор данных является уже отсоединенным от базы данных, можно повторно просматривать элементы данных в любом порядке.

## 2.6. КАК СИНХРОНИЗИРОВАТЬ ИЗМЕНЕНИЯ В НАБОРЕ ДАННЫХ С БАЗОЙ ДАННЫХ

В набор данных можно вносить любые изменения – добавлять новые строки, изменять и удалять существующие строки. Например, добавить строку в таблицу Sessions набора данных можно следующим образом:

```
// создание строки на основе схемы таблицы Sessions набора данных
DataRow newRow = ds.Tables["Sessions"].NewRow();
// получилась пустая строка, столбцы которой должны быть заполнены
newRow["NumGroup"]="903";
newRow["idSubject"]=5;
. . .
// добавление созданной строки в таблицу
ds.Tables["Sessions"].Add(newRow);
```

Однако все изменения происходят с отсоединенным набором данных, т.е. в оперативной памяти компьютера клиентского приложения, но не в базе данных. Чтобы изменения были внесены в исходную базу данных, требуется синхронизировать набор данных и базу с помощью функции класса-адаптера Update(). Однако такая синхронизация возможна только в случае, когда объект-адаптер имеет инициализированные свойства-команды SelectCommand, InsertCommand, UpdateCommand, DeleteCommand. По умолчанию при создании адаптера инициализируется только SelectCommand. Возможно задание остальных команд «вручную».

Для удобства и стандартной инициализации всех свойств-команд адаптера существует специальный класс OdbcCommandBuilder, который генерирует все команды и заполняет соответствующие свойства адаптера. После этого можно проводить синхронизацию изменений с помощью функции Update().

Приведем пример такого программного кода:

```
// создание подключения к базе данных на основе строки соединения
// с указанием источника данных ODBC
OdbcConnection con = new OdbcConnection("DSN=proba");
// подключение к источнику данных
con.Open();
// формирование адаптера для связи с базой данных
OdbcDataAdapter adapt = new OdbcDataAdapter
    ("select * from Sessions", con);
//создание команд для адаптера
OdbcCommandBuilder cb = new OdbcCommandBuilder(adapt);
```

```

// создание набора данных для сохранения результата запроса
DataSet ds = new DataSet();
// заполнение таблицы набора данных (второй параметр)
// из базы данных через адаптер
adapt.Fill(ds, "Sessions");

. . .

// любые действия с набором данных - изменение существующих строк,
// добавление новых, удаление
// синхронизация изменений с базой данных
adapt.Update();
// закрывается соединение
con.Close();

```

## 2.7. ПОЛЬЗОВАТЕЛЬСКИЙ ИНТЕРФЕЙС НА ОСНОВЕ ТАБЛИЦ

Для такого представления данных Windows.Forms имеет специальный элемент управления DataGridView. С помощью этого элемента управления можно добавлять новые записи, изменять значения в ячейках, удалять строки. Программный интерфейс этого класса достаточно сложный, но простые опции можно реализовать минимальными усилиями. Главное условие, чтобы этот элемент управления работал с данными – назначение источника данных для него (свойство DataSource). Кстати, источником данных может быть любая коллекция – список объектов класса, таблица из набора данных и пр.

```

public Form1()
{
    InitializeComponent();
    OdbcConnection con = new OdbcConnection("DSN=proba");
    con.Open();
    OdbcDataAdapter adapt = new OdbcDataAdapter
                                ("select * from Sessions", con);

    ds = new DataSet();
    adapt.Fill(ds, "Sessions");
    con.Close();

    dataGridView1.DataSource = ds.Tables["Sessions"];
}

```



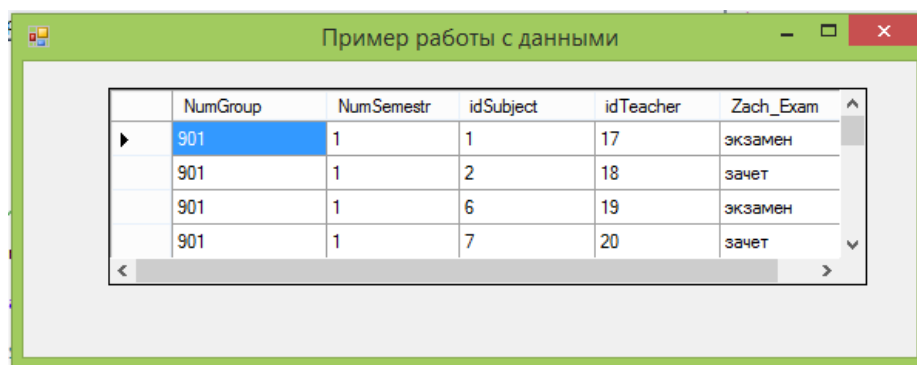


Рис. 42. Загрузка в DataGridView данных из назначенного источника данных.

Однако, как видно из предшествующего рисунка, настройки внешнего вида элемента управления (заголовки столбцов) назначаются автоматически, исходя из схемы таблицы-источника данных.

На следующем рисунке показан интерфейс, который подстроен под пользователя. Для этого требуется провести программную настройку элемента DataGridView – программное создание всех столбцов и привязка их к полям источника данных. Привязка к полям осуществляется с помощью объекта класса BindingSource, который является своеобразным посредником между элементом управления и источником данных.

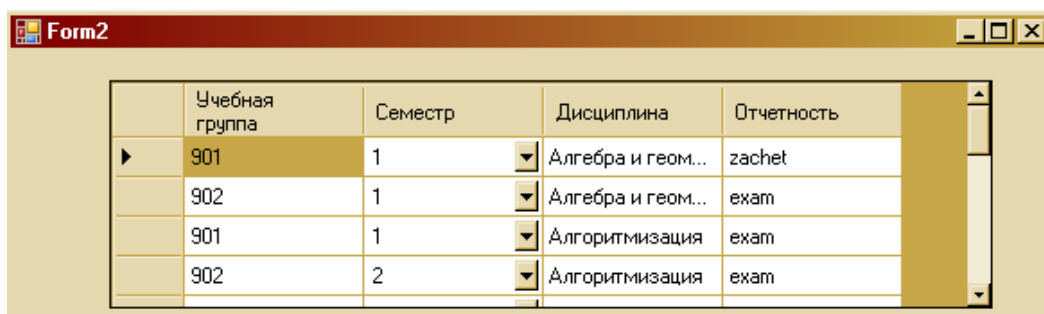


Рис. 43. Вид после программной настройки элемента DataGridView.

```
public partial class Form2 : Form
{
    DataSet ds;
    BindingSource bindsrc = new BindingSource();
    public Form2()
    {
        InitializeComponent();
        OdbcConnection con = new OdbcConnection("DSN=proba");
        con.Open();
        OdbcDataAdapter adapt = new OdbcDataAdapter
            ("select * from Sessions", con);
        ds = new DataSet();
    }
}
```

```

adapt.Fill(ds, "Sessions");
con.Close();

// настройка BindingSource на источник данных -
// таблицу набора данных
bindsrc.DataSource = ds.Tables[0];

// отмена генерации столбцов DataGridView
dataGridView1.AutoGenerateColumns = false;

// установка привязки к источнику данных
dataGridView1.DataSource = bindsrc;

// последовательное создание столбцов элемента управления
DataGridViewTextBoxColumn NumGroup =
    new DataGridViewTextBoxColumn();
// имя поля, которое является источником данных для столбца
NumGroup.DataPropertyName = "NumGroup";
// заголовок столбца
NumGroup.HeaderText = "Учебная группа";
// добавление столбца в коллекцию столбцов DataGridView
dataGridView1.Columns.Add(NumGroup);

DataGridViewComboBoxColumn colSem =
    new DataGridViewComboBoxColumn();
colSem.DataPropertyName = "NumSemestr";
colSem.HeaderText = "Семестр";
colSem.DataSource = new int[] { 1, 2 };
dataGridView1.Columns.Add(colSem);

DataGridViewTextBoxColumn colTitle =
    new DataGridViewTextBoxColumn();
colTitle.DataPropertyName = "idSubject";
colTitle.HeaderText = "Дисциплина";
dataGridView1.Columns.Add(colTitle);

DataGridViewTextBoxColumn colZE =
    new DataGridViewTextBoxColumn();
colZE.DataPropertyName = "Zach_Exam";
colZE.HeaderText = "Отчетность";
dataGridView1.Columns.Add(colZE);
}

```

## 2.8. ПОЛЬЗОВАТЕЛЬСКИЙ ИНТЕРФЕЙС НА ОСНОВЕ ОДНОЙ ЗАПИСИ

Чтобы просмотреть все записи, которые были помещены в результат запроса к базе данных, можно использовать интерфейс последовательного построчного просмотра. Такой интерфейс удобен, например, для приложения тестирования, когда на форме показан только один из вопросов теста.

Для навигации по строкам источника данных можно самостоятельно создать панель инструментов, с помощью которой можно было бы передвигаться по записям, добавлять, изменять и удалять записи источника данных. Но среди доступных элементов управления существует элемент, который уже реализует стандартный вид такой панели. Этот элемент управления называется BindingNavigator и для его корректной работы достаточно настроить только тот же источник данных, что для элементов управления, предоставляющих пользователю данные.

```
public partial class Form3 : Form
{
    DataSet ds;

    // посредник между источником данных и элементами управления
    BindingSource bindsrc = new BindingSource();

    public Form3()
    {
        InitializeComponent();
        OdbcConnection con = new OdbcConnection("DSN=proba");
        con.Open();
        OdbcDataAdapter adapt = new OdbcDataAdapter
                                ("select * from Sessions", con);
        ds = new DataSet();
        adapt.Fill(ds, "Sessions");
        con.Close();

        // настройка источника данных для посредника
        bindsrc.DataSource = ds.Tables[0];

        // настройка источников данных для элементов управления
        // первый параметр - свойство элемента управления,
        // отображающего данные
        // второй параметр - источник данных
        // третий параметр - имя свойства из источника данных,
        // данные которого отображаются в элементе управления
        textBox1.DataBindings.Add("Text", bindsrc, "NumGroup");
        textBox2.DataBindings.Add("Text", bindsrc, "TitleSubject");
        textBox3.DataBindings.Add("Text", bindsrc, "Zach_Ezam");

        // настройка источника данных для панели навигации
        bindingNavigator1.BindingSource = bindsrc;
    }
}
```

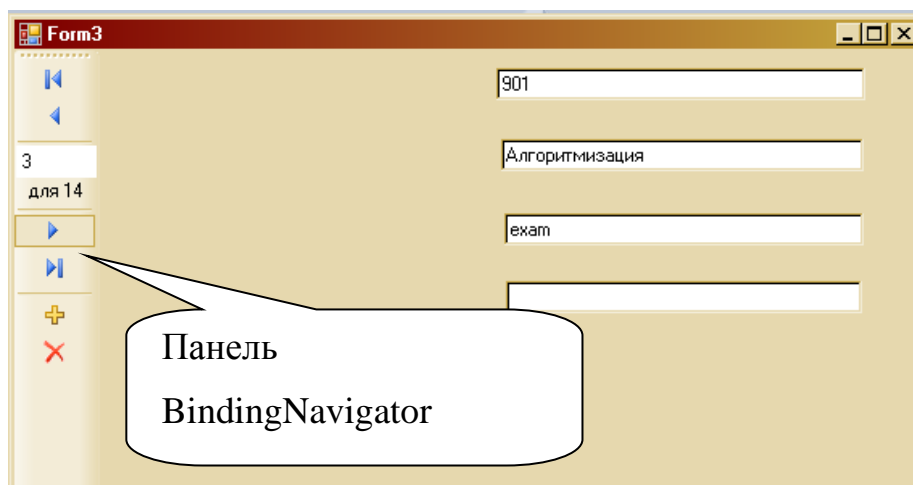


Рис. 44. Вид формы с использованием элемента BindingNavigator.

Отметим, что любой элемент управления на такой форме также должен быть привязан к своему источнику данных, который задается с помощью свойства (коллекции) `DataBindings`. Метод добавления нового источника для элемента управления задается с помощью трех параметров: 1 – свойство элемента управления, которое принимает значение из источника, 2 – объект источника данных или посредника `BindingSource`, 3 – свойство из источника данных, которое отображается в элементе управления.

## 2.9. ГЕНЕРАЦИЯ ОТЧЕТОВ И ПЕЧАТНЫХ ФОРМ

Помимо форм для представления и редактирования информации из базы данных пользователю часто требуется создавать печатные формы, представляющие результаты выполнения сложных запросов. Такие печатные формы принято называть отчетами. Для формирования отчетов в среде Visual Studio существует специальный элемент управления, который называется `ReportView`. Существуют и более сложные дополнительные утилиты (например, не так давно была популярна условно бесплатная утилита `CrystalReports`).

Для добавления в проект отчета требуется установить на форму элемент управления `ReportView`. Далее с помощью конструктора можно сгенерировать новый вид отчета, который будет сохранен в файле с расширением `rdlc`.

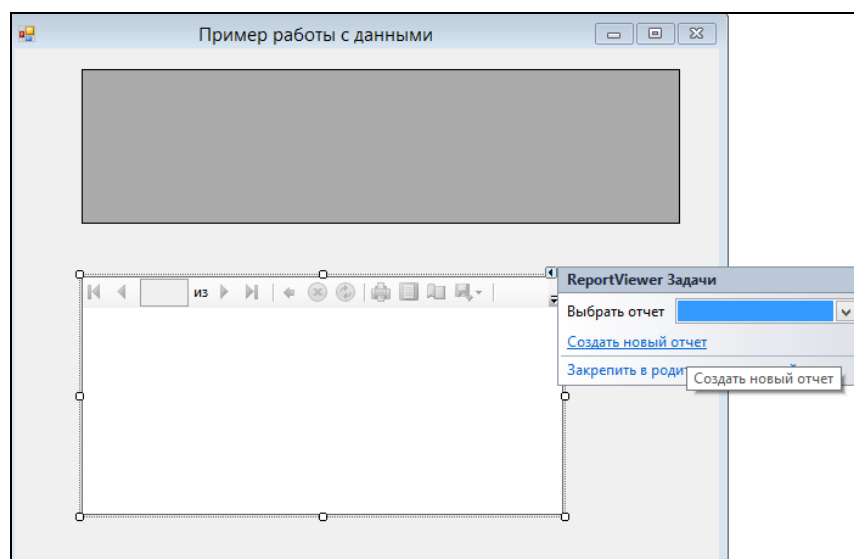


Рис. 45. Вызов конструктора отчета.

Конструктор позволит настроить источники данных, сгенерировать расположение элементов на отчете, задать стилевые и другие характеристики.

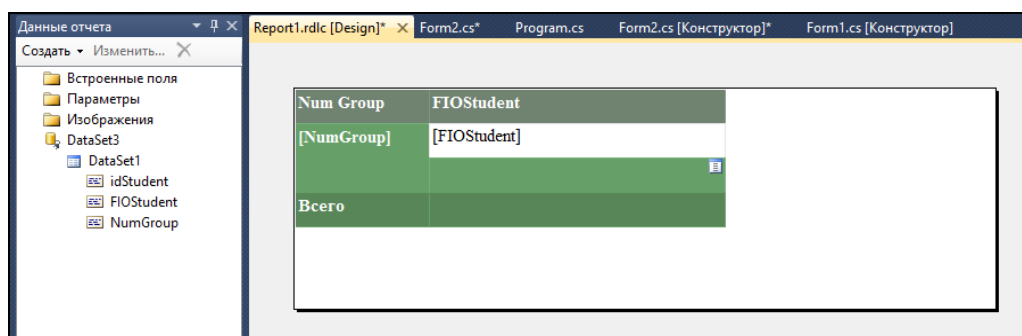


Рис. 46. Вид сгенерированного шаблона отчета.

В сгенерированный отчет можно вносить изменения — добавлять поля, элементы управления, менять стилевые назначения и пр.

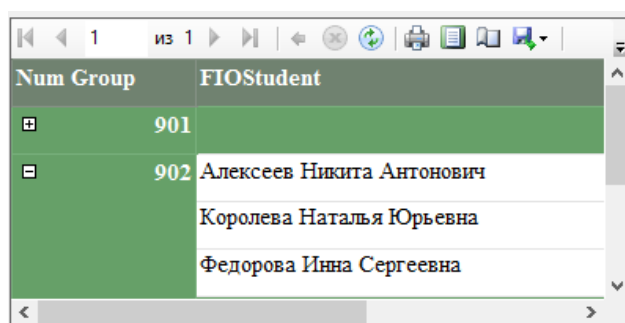
Для использования и показа отчета в элементе управления ReportViewer достаточно сделать несколько простых настроек в программном коде:

```
// указание источника данных для отчета – таблица студентов,
// загруженная в набор данных DataSet
ReportDataSource datasource =
    new ReportDataSource("DataSet1",
        ds.Tables["Students"]);

// настройка на локальный отчет
reportViewer1.ProcessingMode = ProcessingMode.Local;
// очистка старого источника данных отчета из элемента ReportViewer
```

```
reportViewer1.LocalReport.DataSources.Clear();
// настройка пути к файлу с отчетом (.rdlc)
reportViewer1.LocalReport.ReportPath = "../..../Report1.rdlc";
// добавление источника данных (таблицы из набора данных) к отчету
reportViewer1.LocalReport.DataSources.Add(datasource);
// обновление содержимого элемента ReportViewer
reportViewer1.RefreshReport();
```

Результат внедрения в элемент управления отчета может выглядеть следующим образом:



Num Group	FIOStudent
901	Алексеев Никита Антонович Королева Наталья Юрьевна Федорова Инна Сергеевна
902	

Рис. 47. Вид сгенерированного отчета.

## 2.10. ГЕНЕРАЦИЯ ОТЧЕТОВ В ФОРМАТЕ XML

Еще одним способом генерации печатных форм является генерация отчетов в HTML-формате на основе формирования XML-файлов и последующего применения к ним XSLT-преобразований. Этот подход генерации отчетов является более универсальным, так как формат XML и его технологии являются межплатформенной и поддерживаются большим количеством языков программирования и средами проектирования в различных операционных системах.

Например, XSLT-преобразование делается на основе следующего файла. В нем создается список зачетов и экзаменов, которые сдают студенты различных групп. С помощью выражений XPath в тегах `for-each` и `value-of` задаются источники данных из xml-документа для конкретных элементов html-файла, полученного в результате преобразования:

```
<?xml version="1.0" encoding="utf-8"?>
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:msxsl="urn:schemas-microsoft-com:xslt"
  exclude-result-prefixes="msxsl">
  <xsl:template match="/">
```

```

<html>
<head/>
<body>
  <div style="font-family: Book Antiqua;
    font-weight:bold;
    text-align: none;color: #0000FF;">
    Список зачетов и экзаменов.
  </div>
  <ul style="list-style-type: disk">

    <xsl:for-each select="/NewDataSet/Session_Subject">
      <li>
        <xsl:value-of select="NumGroup"/>
        <br/>
        <xsl:value-of select="NumSemestr"/>
        <br/>
        <xsl:value-of select="TitleSubject"/>
        <br/>
        <xsl:value-of select="Zach_Ezam"/>
        <br/>
      </li>

    </xsl:for-each>
  </ul>
</body>
</html>
</xsl:template>
</xsl:stylesheet>

```

Результат применения такого преобразования может быть сохранен в виде html-файла и в дальнейшем загружен в элемент управления WebBrowser, специально предназначенный для работы с файлами такого формата.

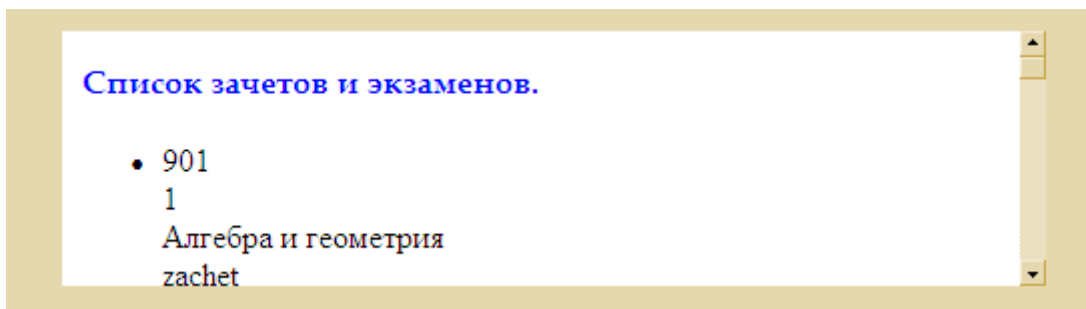


Рис. 48. Вид сгенерированного на основе xslt-преобразования html-документа в элементе управления WebBrowser.

Программный код, который генерирует xml-документ из таблицы набора данных ds, создает на основе xslt-преобразования, которое сохранено в файле forreport.xslt, генерирует html-файл и показывает его в элементе управления WebBrouser.

```

// сохранение набора данных в xml-файле
ds.WriteXml("forreport.xml", XmlWriteMode.WriteSchema);

```

```
// создание специального объекта- xslt-преобразователя
XslCompiledTransform xslt = new XslCompiledTransform();
// загрузка файла с xslt-преобразованием
xslt.Load("../../forreport.xslt");

// выполнение преобразования и генерация на основе xml-файла
//html-файла представления данных
xslt.Transform("forreport.xml", "D:\\forreport.html");

// загрузка полученного html_файла в элемент управления WebBrowser
webBrowser1.Navigate("D:\\forreport.html");
```



### ЧАСТЬ III. ВВЕДЕНИЕ В ХРАНИЛИЩА ДАННЫХ

Хранилища данных – это специальным образом сконструированные базы данных, которые предназначены не столько для хранения информации, сколько для быстрого получения сложных аналитических данных. Перечислим основные характерные моменты, связанные с проектированием и эксплуатацией хранилищ данных, построенных на реляционной модели:

1. Хранилища данных строятся на особой модели базы данных, которая пренебрегает многими аспектами нормализации. В основном, используются схемы типа «Снежинка» и «Звезда». В обеих схемах выделяется центральная **таблица фактов**, которая и содержит данные для анализа, и множество **таблиц измерений** (обычно сильно ненормализованных), содержащие информацию об объектах, в разрезе которых осуществляется анализ, и которые соединены с таблицей фактов связью типа «один-ко-многим».

2. Хранилища не предназначены для большого количества операций модификации данных. Обычно в хранилищах дублируются данные из операционных баз данных (базы, в которые попадают первичные данные). Зато они существенно зависят от операций экспорта из различных источников (не обязательно из баз данных).

3. Большая часть запросов к хранилищу данных связана с таблицей фактов, соединенной только с теми измерениями, которые необходимы для целей анализа.

В качестве учебного примера рассмотрим особенности проектирования и использования хранилища для примера базы с данными об учебном процессе. Для определенности будем использовать в качестве СУБД MS SQL Server. Рассмотрим основные этапы проектирования и использования хранилищ данных.

### 3.1. ПРОЕКТИРОВАНИЕ СХЕМЫ ХРАНИЛИЩА

Рассмотрим схему «Звезда». В качестве таблицы фактов будем использовать таблицу результатов сдачи экзаменов студентами. В качестве таблиц измерений будут фигурировать таблицы «Студенты», «Преподаватели», «Дисциплины».

Итак, у нас получилась довольно простая модель данных с тремя таблицами измерений. Отметим денормализацию на уровне таблицы измерений «Преподаватели» (в нее включается название кафедры, а не ее номер, как это было ранее). Кроме того, отметим наличие еще одного измерения, для которого таблица не создается. Этим измерением является дата сдачи экзамена. Фактами же в данном случае являются набранные баллы и соответствующая им оценка.

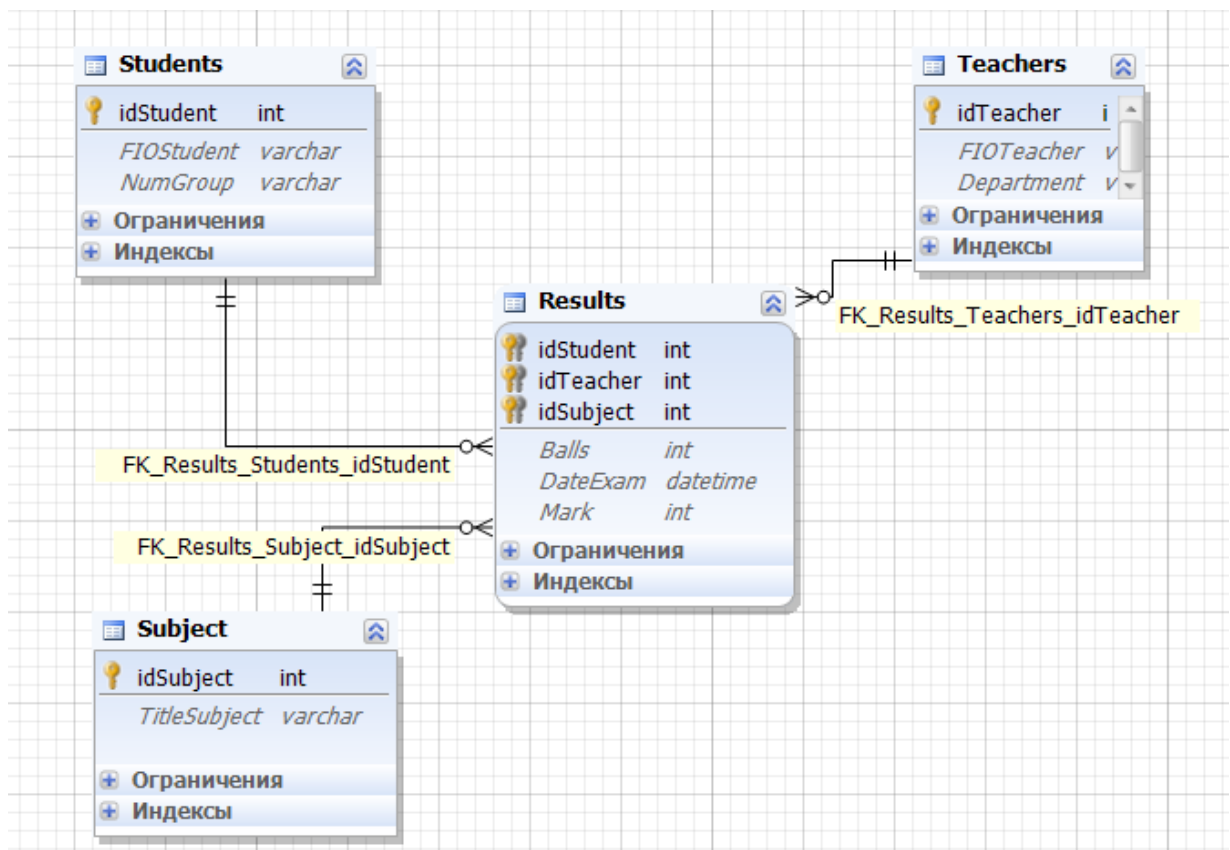


Рис. 49. Схема «Звезда» для хранилища данных.

### 3.2. ЗАГРУЗКА ДАННЫХ

Произведем загрузку данных из базы данных через файлы различного текстового формата.

Начнем с загрузки данных в таблицы измерений.

**Таблица «Students».** Таблица студентов должна быть загружена в том же виде, что и хранится в базе данных деканата. Используем для загрузки файл формата CSV. Этот формат определяет текстовый файл, в котором в первой строке задаются имена столбцов, а каждая следующая строка содержит данные об одной записи, поля разделяются с помощью специального символа-разделителя.

Чтобы создать такой файл оболочка dbForge Studio содержит специальный конструктор экспорта данных. Доступ к нему можно получить, например, с помощью вкладки «Миграция данных» и опции «Экспорт данных». При экспорте придется указать источник данных для экспорта (таблицы или представления) и файл, в который произойдет сохранение информации. В результате будет получен следующий файл:

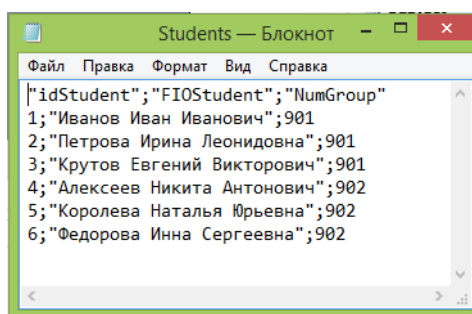


Рис. 50. Файл «Student.csv».

При импорте данных в базу хранилища данных следует воспользоваться той же вкладкой «Миграция данных» и опцией «Импортировать внешние данные». Сначала потребуется выбрать формат и файл с данными, затем определить таблицу, в которую осуществляется импорт, далее будет показан источник данных и можно будет сделать необходимые настройки (пропустить строки,

установить символы-разделители), далее устанавливается соответствие между столбцами источника и приемника данных (в нашем случае они называются одинаково) и далее следуют уточнения по поводу операций импорта (какие операции произвести – добавление новых записей, модификация старых), интерпретация типов данных и прочее.

**Таблица «Subjects».** Таблица дисциплин также должна быть загружена в том же виде. Используем для ее загрузки файл формата XML. Этот формат определяет текстовый файл, в котором структура и данные размечены с помощью специальных тегов. Как и в предыдущем случае следует использовать опции «Экспорт данных» и «Импортировать внешние данные». В результате будет сгенерирован следующий файл:

```
<?xml version="1.0" encoding="utf-8" standalone="yes"?>
<Root xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <xs:schema id="Root"
    xmlns=""
    xmlns:xs="http://www.w3.org/2001/XMLSchema"
    xmlns:msdata="urn:schemas-microsoft-com:xml-msdata">
    <xs:element name="Root"
      msdata:IsDataSet="true"
      msdata:UseCurrentLocale="true">
      <xs:complexType>
        <xs:choice minOccurs="0" maxOccurs="unbounded">
          <xs:element name="Table">
            <xs:complexType>
              <xs:attribute name="idSubject" type="xs:int" />
              <xs:attribute name="TitleSubject" type="xs:string" />
            </xs:complexType>
          </xs:element>
        </xs:choice>
      </xs:complexType>
    </xs:element>
  </xs:schema>
  <Table idSubject="1" TitleSubject="Математический анализ" />
  <Table idSubject="2" TitleSubject="Алгебра и геометрия" />
  <Table idSubject="3" TitleSubject="Теория вероятностей" />
  <Table idSubject="4" TitleSubject="Методы оптимизации" />
  <Table idSubject="5" TitleSubject="Вычислительная математика" />
  <Table idSubject="6" TitleSubject="Алгоритмизация" />
  <Table idSubject="7" TitleSubject="Программирование" />
  <Table idSubject="8" TitleSubject=
    "Объектно-ориентированное программирование" />
  <Table idSubject="9" TitleSubject="Базы данных" />
  <Table idSubject="10" TitleSubject="Web-программирование" />
</Root>
```

**Таблица «Teachers».** При экспорте таблицы преподавателей мы должны на самом деле экспортировать результат его естественного соединения с таблицей кафедр. Для этого придется создать специальное представление данных, так как запрос не может быть использован в качестве источника данных для экспорта.

```
CREATE VIEW Teacher_for_export
AS
SELECT idTeacher, FIOTeacher, TitleDepartment FROM Teachers
INNER JOIN Departments
ON Teachers.idDepartment=Departments.idDepartment;
```

Используем для загрузки данных о преподавателях файл формата Excel, указав при экспорте в качестве источника данных созданное представление. При импорте этих данных следует обратить внимание на настройки относительно строки с заголовками столбцов – требуется явно указать, что первая строка является строкой с именами столбцов, тогда данные будут считываться только со следующей строки. Не забудьте при импорте настроить соответствие для имен столбцов с названием кафедры.

	A	B	C
1	idTeacher	FIOTeacher	TitleDepartment
2	17	Федосеев Александр Иванович	Кафедра математики
3	18	Александрова Анна Петровна	Кафедра математики
4	19	Кириллов Дмитрий Викторович	Кафедра информатики
5	20	Никитин Владимир Иванович	Кафедра информатики
6			

Рис.51. XLS-файл с данными о преподавателях.

Отметим, что экспортировать данные можно и в другие форматы: RTF, PDF и т.д., но для импорта в другую базу эти форматы не будут пригодными.

Вполне естественно, что файл для импорта не обязательно должен быть сгенерирован каким-либо СУБД. Так как таблицу результатов сдачи зачетов-экзаменов в базе данных мы практически не заполняли, экспортировать в таблицу фактов нам пока нечего.

Напишем приложение, которое генерирует файл с оценками, и далее импортируем его в нашу базу данных. Для простоты не будем рассматривать ситуацию, когда разные группы должны сдавать разный набор дисциплин. Будем предполагать, что у каждого студента имеются оценки по всем дисциплинам, которые есть в таблице измерений.

Программный код такого проекта по генерации данных будет следующим. Комментариев в коде должно быть достаточно для понимания алгоритма генерации:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Data;
using System.Data.Odbc;
using System.IO;

namespace generator
{
    class Program
    {
        // в набор данных будут загружены измерения
        // для правильной генерации кодов объектов
        static DataSet ds = new DataSet();

        // вспомогательная функция перевода баллов в оценку
        static int GetMark(int b)
        {
            if (b < 55)
                return 2;
            if (b < 71)
                return 3;
            if (b < 86)
                return 4;
            return 5;
        }

        static void Main(string[] args)
        {
            OdbcConnection con = new OdbcConnection("DSN=proba");
            con.Open();

            // загрузка таблицы студентов
            OdbcDataAdapter adapt1 =
                new OdbcDataAdapter("select * from Students", con);
            adapt1.Fill(ds, "Students");

            // загрузка таблицы дисциплин
            OdbcDataAdapter adapt2 =
                new OdbcDataAdapter("select * from Subjects", con);
            adapt2.Fill(ds, "Subjects");
        }
    }
}
```

```

// загрузка таблицы преподавателей
OdbcDataAdapter adapt3 =
    new OdbcDataAdapter("select * from Teachers", con);
adapt3.Fill(ds, "Teachers");

con.Close();

//создание текстового файла для записи
//сгенерированных данных
StreamWriter tf = new StreamWriter
    (new FileStream("Results.csv", FileMode.Create));

// запись строки заголовка таблицы.
// Для вывода кавычек используется \"
tf.WriteLine("\"idStudent\";\"idSubject\";
    \"idTeacher\";\"DateExam\";
    \"Balls\";\"Mark\"");

// вызов функции генерации данных
GenerateData(tf);

// закрытие файла
tf.Close();
}

// функция генерации оценок экзаменов по всем дисциплинам
static void GenerateData(StreamWriter tf)
{
    Random r = new Random();

    // цикл перебора всех дисциплин
    foreach (DataRow dr_subj in ds.Tables["Subjects"].Rows)
    {
        // генерируем преподавателя,
        // которому сдается данный предмет
        int i = r.Next(ds.Tables["Teachers"].Rows.Count);
        int nom_teach =
            (int)ds.Tables["Teachers"].Rows[i]["idTeacher"];

        //генерируем три даты для сдачи экзамена
        DateTime [] dates=new DateTime[3];

        //определяем зимнюю или весеннюю сессии
        if (r.Next(100) % 2 == 0)
        {
            // зимняя сессия
            dates[0] =
                new DateTime(2014, 1, 5 + r.Next(20));
            dates[1] =
                new DateTime(2014, 1, 5 + r.Next(20));
            dates[2] =
                new DateTime(2014, 1, 5 + r.Next(20));
        }
        else
        {
            // весенняя сессия
            dates[0] =

```

$$\left. \begin{array}{l} \{ \\ \{ \\ \{ \end{array} \right\}$$

базу данных для организации хранилища:

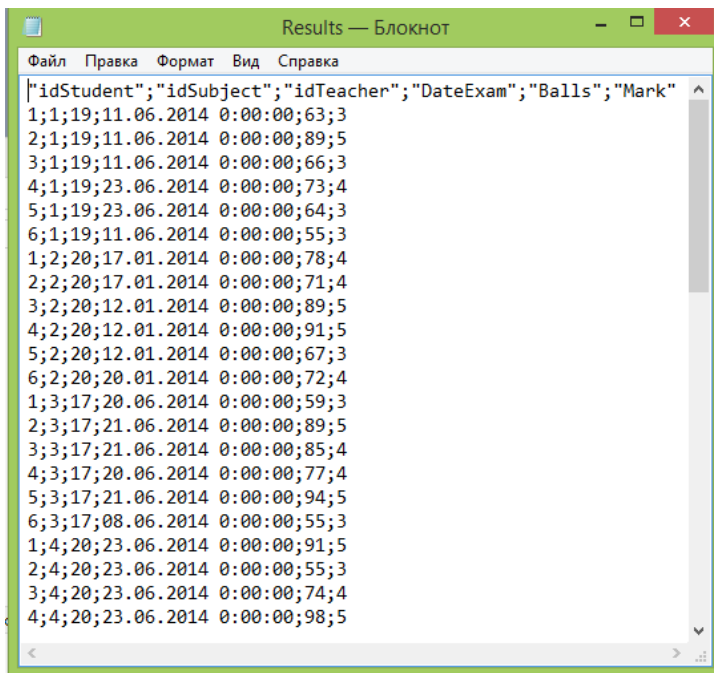


Рис. 52.CSV-файл с оценками студентов.



### 3.3. ПОИСК ИНФОРМАЦИИ В ХРАНИЛИЩЕ

По своей сути таблица фактов представляет собой OLAP-куб (Online Analytical Processing), который позволяет быстро находить интересующую информацию в разных аспектах. Существуют стандартные операции с OLAP-кубами: срез, вращение, консолидация и детализация. Кратко опишем суть этих операций:

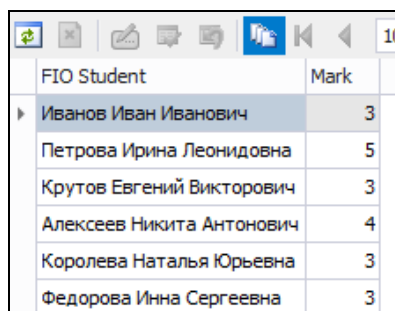
- Операция среза обычно определяет все возможные данные с фиксацией одного или нескольких измерений.
- Операция вращения предполагает переупорядочивание измерений.
- Операция консолидации предполагает выполнение агрегирования данных по отдельным измерениям.
- Операция детализации предполагает получение нового куба, содержащего не все значения измерений.

А теперь продемонстрируем на примере ряда запросов применение этих операций для получения информации. Обычно все запросы к хранилищу данных обращаются к таблице фактов или к естественному соединению таблицы фактов с необходимыми для запроса таблицами измерений:

- Получить все оценки по дисциплине «Математический анализ». В этом случае мы используем операцию среза. Можно сказать, что это одновременно является срезом по измерению «Преподаватель» (так как согласно генерации данных дисциплина ведется только одним преподавателем):

```
SELECT FIOStudent, Mark FROM Results INNER JOIN Students
    ON Results.idStudent=Students.idStudent
    WHERE idSubject =
        (SELECT idSubject FROM Subjects
            WHERE TitleSubject='Математический анализ');
```

Результатом является другой гиперкуб, в данном случае с одним измерением FIOStudent и фактом - оценкой.

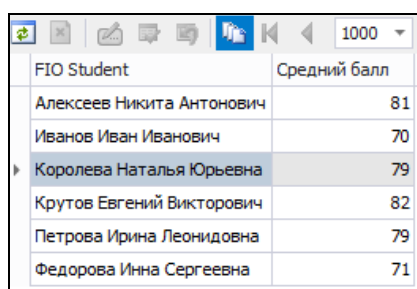


FIO Student	Mark
Иванов Иван Иванович	3
Петрова Ирина Леонидовна	5
Крутов Евгений Викторович	3
Алексеев Никита Антонович	4
Королева Наталья Юрьевна	3
Федорова Инна Сергеевна	3

Рис. 53. Оценки студентов по дисциплине «Математический анализ».

- Получить средние баллы всех студентов. Здесь демонстрируется операция консолидации, которая выражается укрупнением группировок по измерениям.

```
SELECT FIOStudent, AVG(Balls) AS "Средний балл" FROM
Results INNER JOIN Students
ON Results.idStudent=Students.idStudent
GROUP BY FIOStudent;
```

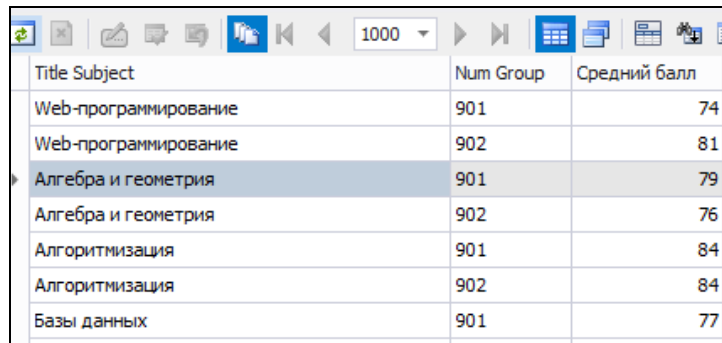


FIO Student	Средний балл
Алексеев Никита Антонович	81
Иванов Иван Иванович	70
Королева Наталья Юрьевна	79
Крутов Евгений Викторович	82
Петрова Ирина Леонидовна	79
Федорова Инна Сергеевна	71

Рис. 54. Средние баллы студентов.

- Получить средние баллы сдачи экзаменов по группам студентов. В данном случае производится консолидация по более крупной группе – номерам групп студентов. Таким образом, результатом является гиперкуб, в котором два измерения – дисциплина и номер учебной группы. В каком-то смысле здесь производится переупорядочивание измерений, поэтому можно говорить и о применении операции вращения.

```
SELECT TitleSubject, NumGroup, AVG(Balls) AS "Средний балл" FROM
Results INNER JOIN Subjects
ON Results.idSubject=Subjects.idSubject
INNER JOIN Students ON Results.idStudent=Students.idStudent
GROUP BY TitleSubject, NumGroup ORDER BY TitleSubject;
```

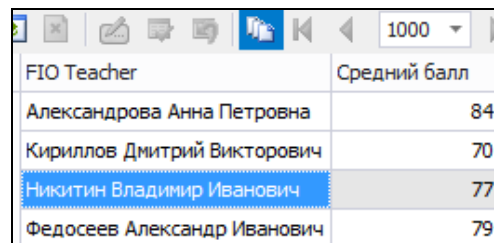


Title Subject	Num Group	Средний балл
Web-программирование	901	74
Web-программирование	902	81
Алгебра и геометрия	901	79
Алгебра и геометрия	902	76
Алгоритмизация	901	84
Алгоритмизация	902	84
Базы данных	901	77

Рис. 55. Средние баллы по предметам и группам.

- Получить средние баллы по преподавателям. Какому студенту не хотелось знать перед экзаменом, какой преподаватель более лояльный? Ситуация аналогичная запросу о средних баллах студентов. Как видно, нашелся один заметно более строгий преподаватель.

```
SELECT FIOTeacher, AVG(Balls) AS "Средний балл" FROM
    Results INNER JOIN Teachers
    ON Results.idTeacher=Teachers.idTeacher
    GROUP BY FIOTeacher;
```

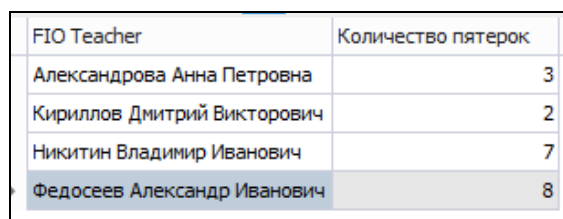


FIO Teacher	Средний балл
Александрова Анна Петровна	84
Кириллов Дмитрий Викторович	70
Никитин Владимир Иванович	77
Федосеев Александр Иванович	79

Рис. 56. Средние баллы по преподавателям.

- Получить количество пятерок, которые поставили преподаватели.

```
SELECT FIOTeacher, COUNT(Mark) AS "Количество пятерок" FROM
    Results INNER JOIN Teachers
    ON Results.idTeacher=Teachers.idTeacher
    WHERE Mark=5
    GROUP BY FIOTeacher;
```

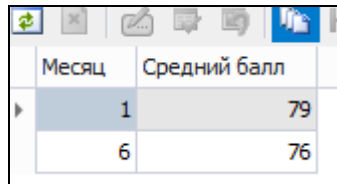


FIO Teacher	Количество пятерок
Александрова Анна Петровна	3
Кириллов Дмитрий Викторович	2
Никитин Владимир Иванович	7
Федосеев Александр Иванович	8

Рис. 57. Количество пятерок, которые поставили преподаватели.

- Узнать, в какую сессию (зимнюю или летнюю) студенты сдают экзамены лучше. Для этого мы рассчитаем средние баллы сдачи экзаменов зимой и летом. Здесь мы опять сталкиваемся с операцией консолидации, но по нестандартному измерению – дате сдачи экзамена.

```
SELECT DATEPART(month, DateExam) AS "Месяц",
       AVG(Balls) AS "Средний балл"
FROM Results GROUP BY DATEPART(month, DateExam);
```

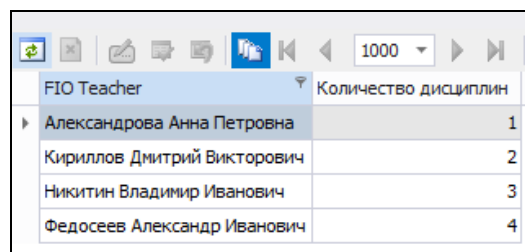


Месяц	Средний балл
1	79
6	76

Рис. 58. Средние баллы в зимнюю и летнюю сессии.

- Получить количество дисциплин, которые ведет каждый из преподавателей. Здесь в качестве факта выступает измерение таблицы фактов idSubject. В данном случае сначала применяем операцию детализации, выбирая для нового гиперкуба из таблицы фактов только некоторые данные, после чего применяется операция консолидации посредством группировки по измерению «Преподаватель».

```
SELECT FIOTeacher, COUNT(idSubject) AS "Количество дисциплин" FROM
  (SELECT DISTINCT FIOTeacher, idSubject FROM Results
   INNER JOIN Teachers
   ON Results.idTeacher=Teachers.idTeacher) AS Q1
GROUP BY Q1.FIOTeacher;
```



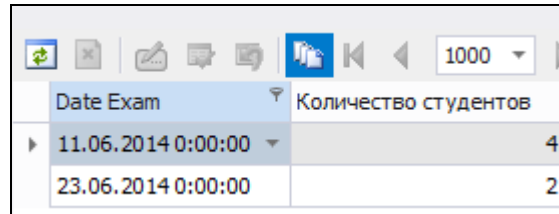
FIO Teacher	Количество дисциплин
Александрова Анна Петровна	1
Кириллов Дмитрий Викторович	2
Никитин Владимир Иванович	3
Федосеев Александр Иванович	4

Рис. 59. Количество дисциплин каждого из преподавателей.

- Получить количество студентов, сдавших дисциплину «Математический анализ» на каждую дату сдачи этого предмета. Здесь используется опера-

ция среза, после чего применяется операция консолидации по дате сдачи экзамена.

```
SELECT DateExam, COUNT(*) AS "Количество студентов" FROM Results
WHERE idSubject=(SELECT idSubject FROM Subjects WHERE
TitleSubject='Математический анализ')
GROUP BY DateExam;
```

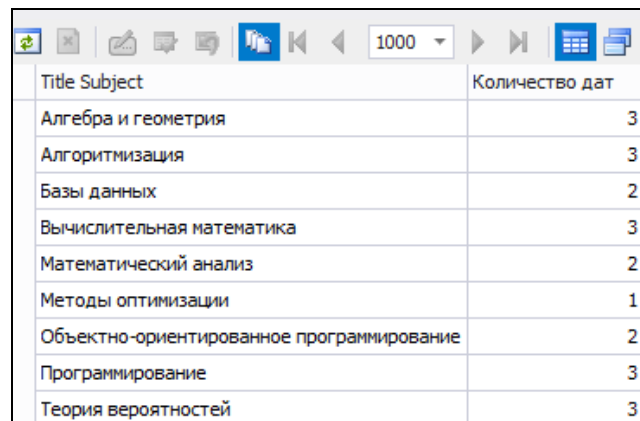


Date Exam	Количество студентов
11.06.2014 0:00:00	4
23.06.2014 0:00:00	2

Рис. 60. Количество студентов, сдавших математический анализ на разные даты.

- Получить количество дат, в которые проводились экзамены по каждому из предметов. Данный запрос будет связан с последовательностью операции детализации и консолидации.

```
SELECT TitleSubject, COUNT(DateExam) AS "Количество дат" FROM
(SELECT DISTINCT TitleSubject, DateExam FROM Results
INNER JOIN Subjects
ON Results.idSubject=Subjects.idSubject) AS Q1
GROUP BY Q1.TitleSubject;
```



Title Subject	Количество дат
Алгебра и геометрия	3
Алгоритмизация	3
Базы данных	2
Вычислительная математика	3
Математический анализ	2
Методы оптимизации	1
Объектно-ориентированное программирование	2
Программирование	3
Теория вероятностей	3

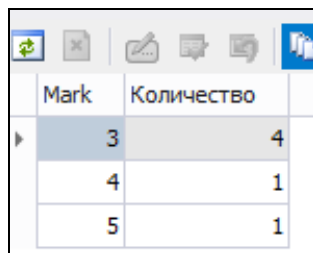
Рис. 61. Количество дат сдачи каждого из экзаменов.

- Получить количество оценок каждого типа по дисциплине «Математический анализ». В данном примере применение среза и последующей детализации приведет к построению гиперкуба, в котором можно сделать факт (оценку) измерением. Затем к этому новому гиперкубу применяется операция консолидации.

```

SELECT Mark, COUNT(*) AS "Количество" FROM Results
WHERE idSubject =
      (SELECT idSubject FROM Subjects
       WHERE TitleSubject='Математический анализ')
GROUP BY Mark;

```



Mark	Количество
3	4
4	1
5	1

Рис. 62. Количество оценок каждого типа по математическому анализу.

### 3.4. ПОСТРОЕНИЕ ОТЧЕТОВ С ПОМОЩЬЮ ЗАПРОСОВ К ХРАНИЛИЩУ

Оболочка dbForge Studio содержит собственные средства для создания отчетов. Отчеты формируются в виде файлов с расширением rbd. Они содержат разметку элементов данных в отчете. Файл можно создать с помощью конструктора, который вызывается с помощью вкладки «Анализ данных» стартовой страницы и пункта «Дизайн нового отчета».

На первой странице дизайнера (мастера) предлагается выбрать базу данных и источник данных (таблицу или запрос). На следующей странице можно будет выбрать таблицу или написать текст запроса для отображения в отчете. Далее можно будет выбрать поля для отображения в отчете. На следующем шаге задаются принципы группировки записей (группировок может быть несколько и они могут образовывать иерархию). Далее следуют несколько шагов с заданием характеристик внешнего вида отчета.

Например, на основании запроса, представляющего собой соединение таблицы фактов и всех таблиц измерений, сгенерируем отчет:

Стартовая страница | Отчёт 1.rbd

1 . . . . . 1 . . . . . 2 . . . . . 3 . . . . . 4 . . . . . 5 . . . . . 6 . . . . .

reportHeaderBand1 [один раз в отчёте]

Отчет по оценкам

pageHeaderBand1 [один раз на странице]

Num Group	Title Subject	FIOTeacher	Date Exam	Balls	Mark	FIOStudent
-----------	---------------	------------	-----------	-------	------	------------

groupHeaderBand1

[NumGroup]

groupHeaderBand2

[TitleSubject] [FIOTeacher]

groupHeaderBand3

[DateExam]

Detail

[Balls] [Mark] [FIOStudent]

pageFooterBand1 [один раз на странице]

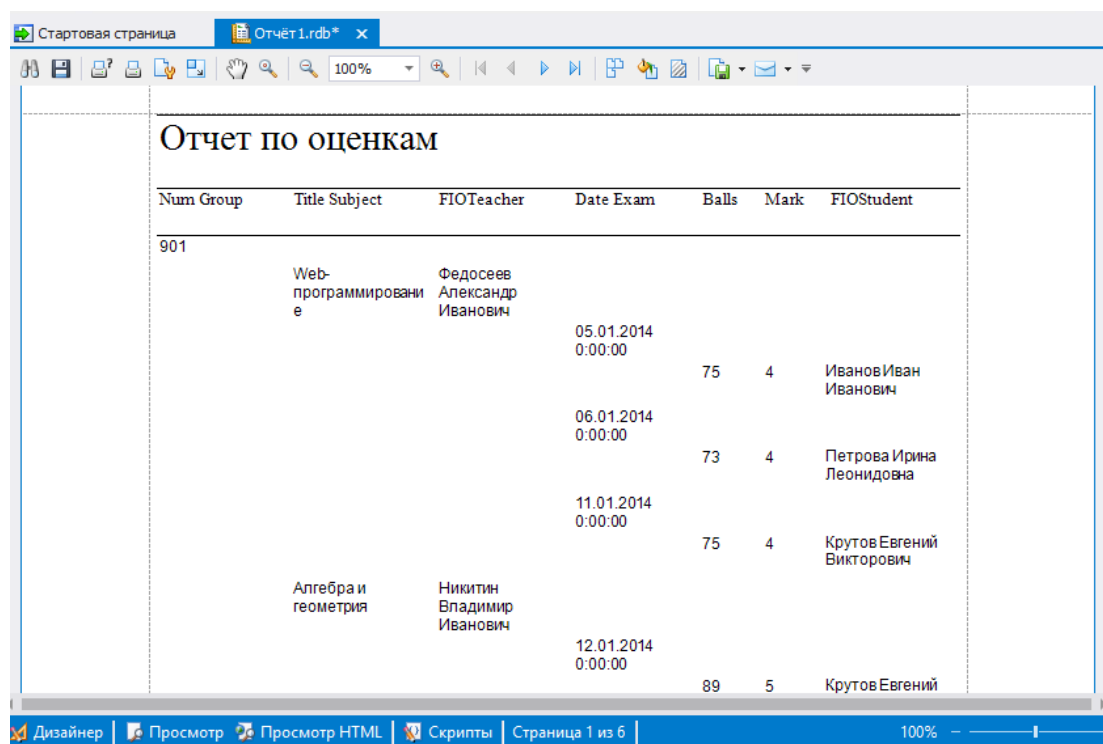
7 августа 2014 г. Page 1 of 1

Рис. 63. Вид отчета.

Основная группировка данных производится по номеру группы, далее по иерархии идут группировки по дисциплине (преподавателю), дате экзамена. Детализацией записи являются ФИО студента, набранные баллы и полученная оценка.

Сгенерированный отчет можно редактировать с помощью специальных панелей и окон свойств.

Результат генерации отчета можно посмотреть, войдя в режим просмотра (вкладка внизу области отчета).



Num Group	Title Subject	FIOTeacher	Date Exam	Balls	Mark	FIOStudent
901	Web-программирование	Федосеев Александр Иванович	05.01.2014 0:00:00	75	4	Иванов Иван Иванович
			06.01.2014 0:00:00	73	4	Петрова Ирина Леонидовна
			11.01.2014 0:00:00	75	4	Крутов Евгений Викторович
	Алгебра и геометрия	Никитин Владимир Иванович	12.01.2014 0:00:00	89	5	Крутов Евгений

Рис. 64. Сгенерированный отчет.



## СПИСОК ЛИТЕРАТУРЫ

1. Бейли, Л. Изучаем SQL [Текст]: пер.с англ./ Линн Бейли – СПб: ИД «Питер», 2012. – 592 с.
2. Кренке, Д. Теория и практика построения баз данных [Текст]: пер.с англ. / Дэвид Кренке. – СПб: Питер, 2003. – 800 с.
3. Ульман, Д. Основы реляционных баз данных [Текст]: пер.с англ. / Джеффри Д.Ульман, Дженифер Уидом. – М: Вильямс, 2006. – 382 с.
4. Лобел, Л. Разработка приложений на основе Microsoft SQL Server 2008 [Текст]: пер с англ./ Л. Лобел, Э.Дж. Браст, С. Форте. – СПб.:БХВ-Петербург, 2010. – 1024 с.
5. Кузнецов, М. MySQL 5 в подлиннике [Текст] / Максим Кузнецов, Игорь Симдянов. – СПб: БХВ-Петербург, 2010. – 1007 с.
6. Уорсли, Д. PostgreSQL [Текст]: пер.с англ. /Дж.Уорсли, Дж.Дрейк. – СПб: Питер, 2003. – 496 с.
7. Пинягина, О.В. Практикум по курсу "Базы данных" [Текст]/ О.В.Пинягина, И.А.Фукин. – Казань, Казанский федеральный университет, 2012. – 92 с.
8. Пирогов, В.Ю. Информационные системы и базы данных: организация и проектирование [Текст]/ В.Ю.Пирогов. – СПб: БХВ-Петербург, 2009. – 528 с.
9. Советов, Б.Я. Базы данных: теория и практика: Учебник для бакалавров [Текст]/ Б.Я. Советов, В.В. Цехановский, В.Д. Чертовской. – М.: Юрайт, 2013. – 463 с.

*Учебное пособие*

**Андрианова Анастасия Александровна**  
**Мухтарова Татьяна Маратовна**  
**Рубцова Рамиля Гакилевна**

**ЛАБОРАТОРНЫЙ ПРАКТИКУМ ПО КУРСУ  
«ТЕХНОЛОГИИ БАЗ ДАННЫХ»**

Дизайн обложки  
***Р.Г.Рубцова***

Подписано в печать \_10.03.2016.  
Бумага офсетная. Печать цифровая.  
Формат 60х84 1/16. Гарнитура «Times New Roman». Усл. печ. л. .  
Тираж экз. Заказ

Отпечатано с готового оригинал-макета  
в типографии Издательства Казанского университета

420008, г. Казань, ул. Профессора Нужина, 1/37  
тел. (843) 233-73-59, 233-73-28